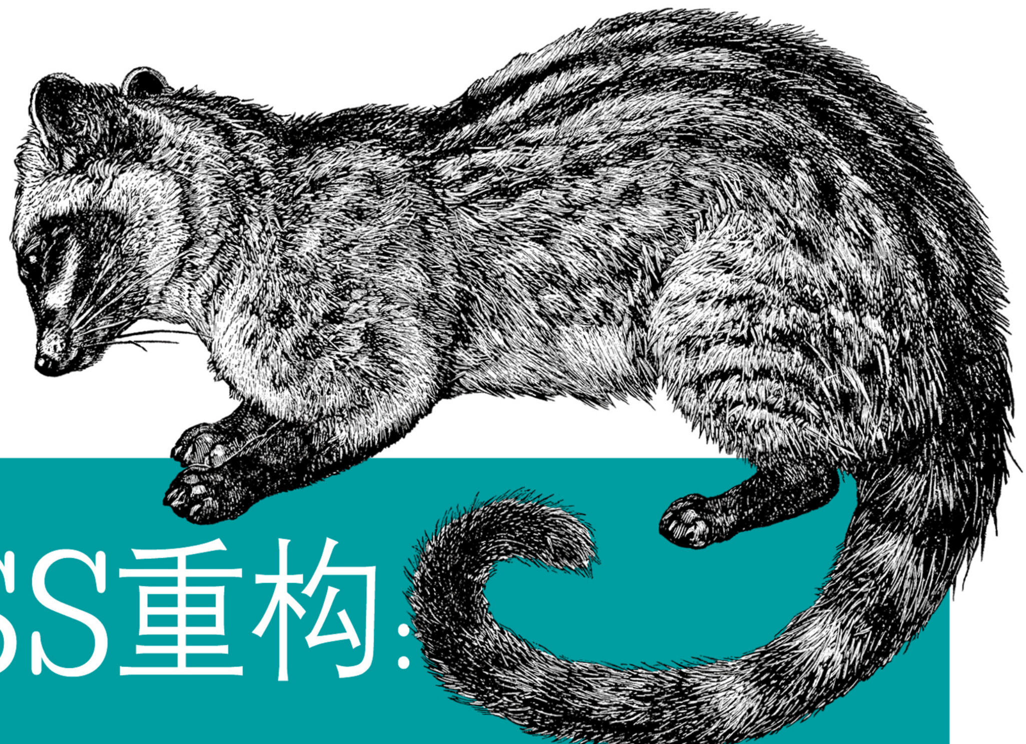
O'REILLY

TURING 图灵程序设计丛书



CSS重构：样式表性能调优

CSS Refactoring: Architect Your Stylesheets for Success

探索如何编写结构合理的CSS，通过重构让代码性能更高、更易于维护

[美] Steve Lindstrom 著

杜春晓 司韦韦 译

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

译者介绍

杜春晓

英语语言文学学士，软件工程硕士。译有《Python数据挖掘入门与实践》《JavaScript之美》等书。宜_生@新浪微博。

司韦韦

前端工程师，曾任职于敦煌网、中科汇联。

TURING 图灵程序设计丛书

CSS重构：样式表性能调优

CSS Refactoring:
Architect Your Stylesheets for Success

［美］Steve Lindstrom 著
杜春晓 司韦韦 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（CIP）数据

CSS重构 : 样式表性能调优 / (美) 史蒂夫·林德斯特伦 (Steve Lindstrom) 著 ; 杜春晓, 司韦韦译. -- 北京 : 人民邮电出版社, 2017.11
（图灵程序设计丛书）
ISBN 978-7-115-46978-6

Ⅰ. ①C… Ⅱ. ①史… ②杜… ③司… Ⅲ. ①网页制作工具 Ⅳ. ①TP393.092.2

中国版本图书馆CIP数据核字(2017)第243193号

内 容 提 要

本书作为 CSS 重构指南，不仅展示了如何编写结构合理的 CSS，以构建响应式、易于使用的网站，还介绍了如何用重构方法创建可读性更强和更易于维护的 CSS 代码。

本书适合所有 CSS 开发人员。

◆ 著　　[美] Steve Lindstrom
译　　杜春晓　司韦韦
责任编辑　朱　巍
执行编辑　张海艳
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：8.75
字数：207千字　　　　2017年 11 月第 1 版
印数：1－3 500册　　　2017年 11 月北京第 1 次印刷
著作权合同登记号　图字：01-2017-6480号

定价：39.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

谨以此书献给与我同甘共苦的家人；没有你们的支持，我不可能取得今天这样的成就。

目录

译者序

人们常将前端技术 HTML、CSS 和 JavaScript 亲切地称为“三剑客”，大抵说的是它们并肩支撑起了数以亿计的网站，英勇、侠义之气概不亚于桃园结义的刘关张。但若将网站比作一个少女，HTML 可视作她的身躯，CSS 必然是伊人的着装打扮，而 JavaScript 则是其言谈举止。胭脂铅粉易得，化妆技巧难学。CSS 何尝不是这样一门技术，稍微有点 CSS 知识的前端新人，都可以借助可视化编辑工具，随心所欲地加一串串行内样式，实现自己想要的效果。如此看来，CSS 很简单。但实际上，CSS 又很难，后台开发人员看到前端高手写的一堆堆代码就头大，觉得 CSS 好难啊！改动一处样式，结果页面布局大乱，修改起样式来如履薄冰，这哪里是哪里呀！样式明明加上了，可千呼万唤就是不生效！伺候好一个浏览器，别家的又乱了，这还能按时发布吗？！

CSS 的难，部分原因正是其简单造成的。代码写起来简单、灵活，且有近乎万能的行内样式，实在不行还有更厉害的 `!important` 声明，所以可以尽管“大胆”地写：定义冗长的选择器，大段粘贴样式，CSS 和 JavaScript 中混用选择器，弃而不用的代码也不删掉，不同用途的样式混在一起。虽然应用样式的目的达到了，但是代码的可读性、性能疏于考虑，华丽的外表下惨不忍睹。

想必很多前端朋友都已经认识到上述问题，并开始有意识地积累 CSS 重构知识。本书刚好适合你。本书是 CSS 重构这个垂直领域的系统指南，作者 Steve Lindstrom 有近二十年的网站开发经验。该书首先从软件架构的高度和软件工程的视角介绍了 CSS 重构的必要性、重构的原则和重构时间点的选择，接着讲解了级联方法、选择器特指度的计算方法以及 CSS 编码规范，然后又介绍了如何测试网站在多种设备和浏览器上的展示效果。你将学到如何用 Gemini 测试框架、PhantomJS 无头浏览器自动化截图，比较重构前后视觉效果上的差异。最后，作者详细介绍了代码的组织方式、重构策略以及重构的评价标准。读完本书，始可与言 CSS 重构已矣。从这本薄书中你收获到的将是一整套 CSS 重构理论加方法指南。此后，你可将自己已掌握和新掌握的知识纳入这个体系，以重构的理念贯穿始终，构建起自己的前端知识大厦，最终输出高性能的 CSS 代码。本书适合编写 CSS 的前端开发人员，

新手宜于学习、实践和佐证自己的想法，高手可当是相互切磋。

我周围有很多前端工程师朋友，工作中也会与他们打交道。他们或为网站，或为 Android、iOS 应用编写 CSS 样式。我知道有不少前端朋友之前从事其他行业，从培训班毕业后直接找的工作。我在此郑重向感觉自己缺乏软件工程知识和实践的朋友推荐本书，书中提到的方法能将你引向大路。互联网技术这个行当相对不那么看重各种背景，只要踏实、上进、肯钻研，再加上技术过硬，都能找到一个好去处。

我跟前端还是蛮有渊源的。十年前，我开始傻乎乎地用记事本写 HTML、CSS，添加幼稚的样式，为实现的朴素效果激动不已。读研期间的一次大作业，我曾选择前端技术作为研究方向。我还曾把翻译 HTML 技术文章作为翻译实践。我妻子司韦韦曾从事前端工作。我见证了她转行学习前端到工作的整个过程，我常常不由地为她的华丽转身而惊叹！她在短短几年之内，从一家小外贸公司的职员蜕变为“宇宙中心”一家知名外贸电商的前端工程师。时代造人，不能不让人感慨。技术发展的脚步，我们不能不察；技术发展的速度之快，我们不能不奋起直追。还记得她刚工作时，一个夏日的周末，我们同去中关村图书大厦读书，然后又一起去北大静园草坪游玩。云销雨霁，我们拣地热井旁的大青石拂去雨水而坐，她兴奋地跟我说，她在制作页面方面的很多想法与刚读到的书中的想法如出一辙，我跟着她一起高兴。一本好书的力量是无穷的。我希望本书所讲内容也能给广大前端工程师带来知识、信心和力量。

感谢作者 Steve Lindstrom 将自己多年总结得来的宝贵经验分享给我们。感谢图灵公司的朱巍等诸位编校人员，本书的顺利出版离不开你们幕后的辛勤付出，感谢谢婷婷编辑在译文语言风格等方面的诸多指导。感谢我的同学肖铮，前端这个职位我第一次是从他那里听来的。感谢通过前端技术结识的同事和朋友，他们分别是张琳琳、王海霞、邵有生、赵伟轩、曹倩倩等。感谢 2009 级计算机辅助翻译专业的同学对我翻译工作的支持，同窗几载，受益甚多。在翻译本书的过程中，女儿有时会顽皮地钻到我面前，一定要敲敲键盘才肯罢休。姑娘，你这是长大了要做前端吗？感谢我的家人，我得以心无旁骛做自己喜欢的事，全仰仗众人的携助。

本人学识有限，且翻译仓促，书中难免会有错误、不当和疏漏之处，敬请读者批评指正。

杜春晓
2017年2月

前言

刚开始学 CSS 那会儿，我发现它的**句法**（组成一门编程语言的规则和结构）掌握起来很简单，因为代码的编写方式有现成的规则可循。然而，我发现组织好 CSS，使其易于维护，则难度比较大。比这更难的是整理之前写的、结构不清晰的 CSS。我写这本书的初衷就是想把自己在试错过程中学到的一切分享给更多人。若是我刚开始学 CSS 那会儿，市面上有这样一本书就好了。

目标读者

虽然我希望所有 CSS 开发人员都能从中受益，但本书主要是写给那些勉强能够编写可用的用户界面的读者的，他们缺乏经验或视野不够开阔，不能从全局理解自己编写的代码是怎么组织在一起的。目标读者懂得怎样编写 CSS 语句，但也许不理解某些做法背后的原因。他们可能也不知道怎么安排代码的架构，使其成为易维护、易扩展并且便于合作开发的软件。

本书的目标

本书旨在阐明 CSS 的一些比较微妙的知识点，便于新手理解。我还想讲讲为什么编写和测试 CSS 很难，以及为什么说花时间重构 CSS 是值得的。

本书主题如下：

- 什么是重构，它的好处是什么，它与软件架构之间什么关系
- 级联、选择器的优先级和盒子模型等常被误解的 CSS 知识
- 如何以更明智的方式编写代码并保持一致性，从而编写出质量更高的 CSS
- 如何用编码标准和模式库维护高质量的 CSS
- CSS 的测试方法
- CSS 的组织方法

- CSS 的重构策略
- 衡量重构是否成功的标准

你可以将本书中的知识立刻付诸实践，去编写质量更高的 CSS 代码，从而提高代码库的质量。倘若团队协同工作，代码库也会因此更容易维护。若在实践过程中用到书中讲的概念，我鼓励你再次阅读相关章节，彻底把它搞清楚。

本书不涉及的内容

本书重点讲解的这些概念，其技术性不一定很强。因此，有很多主题不在本书讲解范围之内，其中包括以下几个。

CSS属性

编写 CSS 需要掌握 CSS 属性的相关知识，但是本书不讲。也许本书会时不时地建议你该使用什么属性，但是为了能够系统地学习这些属性，建议你参考 Eric Meyer 的《CSS 权威指南》、Christopher Schmitt 的《CSS Cookbook 中文版》或其他权威网站，比如 Mozilla Developer Network（https://developer.mozilla.org）。

HTML结构安排

构建用户界面既需要 HTML 又需要 CSS，它们会相互影响。我们会讨论将 CSS 从 HTML 中分离出来的方法，但是不会讨论 HTML 编写方式及其结构安排的利弊。

前端性能

搭建网站，前端性能很重要，这个主题也非常有趣。可是由于本书只讲 CSS 重构，因此我们只是简要讲讲前端性能。这个话题涉及面很广，囊括了其他多个主题。Steve Souders（https://stevesouders.com）写了几本性能方面的书，都非常不错。Paul Irish（http://www.paulirish.com）、Nicole Sullivan（http://www.stubbornella.org）和 Stoyan Stef anov（http://www.phpied.com）三人也在这个方面做出了很多了不起的工作。此外，谷歌提供的一些指南和工具（https://developers.google.com/speed/pagespeed），对于提升前端性能也很有帮助。

CSS框架

CSS 框架变动比较频繁，在实现上强行加入了自己定义的规则，因此本书也不讲。然而，我希望你在读完本书后能够看着任意一个框架的源代码自己去评价一下其实现的好坏。

小众浏览器

Web 浏览器数量惊人，但我们只讨论主流浏览器，比如 Microsoft Edge（之前的 IE）、Safari、Chrome 和 Firefox 以及它们的移动浏览器版本，因为这些浏览器占据了绝大多数市场份额。

术语

本书假定目标读者具备一定的 CSS 知识，但他们对有些术语可能不熟悉，因此书中会对一些术语作出解释。下面这些术语更为基础，先在这里列出来。

- **选择器**是指为一个或一组元素添加样式时所使用的模式。
- **声明块**是一组规则，表示为 HTML 元素应用什么属性，属性取什么值。
- **属性**表明为选中的元素应用什么样式。我们需要为属性指定**取值**。
- **规则集**由一个或多个选择器和一个声明块组成。

例 P-1 的 CSS 代码要求浏览器为所有段落应用蓝色、16px（像素）大小的样式。`p` 为选择器，告诉浏览器为哪个元素添加样式。左右花括号之间的所有内容为声明块。该声明块包括两条声明语句：第一条为 `color` 属性赋值 `#1200FF`，第二条为 `font-size` 属性赋值 `16px`。整个这一段代码即为规则集。为了方便理解，我们在图 P-1 中标明了规则集的各个组成部分。

例 P-1 规则集示例

```
p {
    color: #1200FF;
    font-size: 16px;
}
```

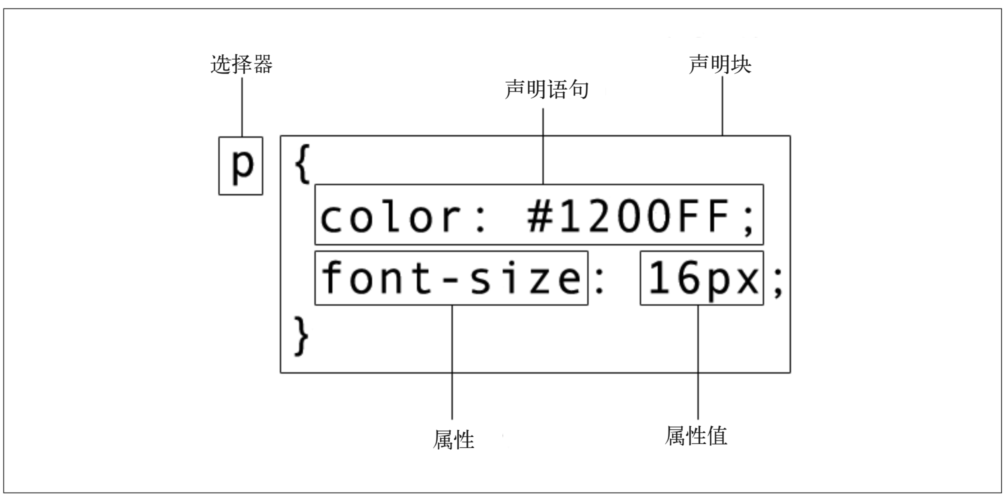


图 P-1：CSS 规则集图解

配套网站提供的内容

本书的配套网站 http://www.cssrefactoringbook.com 提供以下内容：

- 本书各示例的代码文件
- 偶尔会发博客文章
- 好文章、演讲稿和其他资源的链接
- 勘误和纠错信息

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*CSS Refactoring: Architect Your Stylesheets for Success* by Steve Lindstrom (O'Reilly). Copyright 2017 Steve Lindstrom, 978-1-491-90642-2"。

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

排版约定

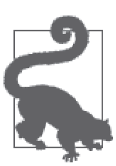
本书使用了下列排版约定。

- **黑体**
  表示新术语或重点强调的内容。

- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。

- 加粗等宽字体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。

- 等宽斜体（*`constant width italic`*）
  表示应该由用户输入的值或根据上下文确定的值替换的文本。



该图标表示提示或建议。



该图标表示一般标注。



该图标表示警告或警示。

O’Reilly Safari

Safari（原来叫 Safari Books Online）是面向企业、政府、教育从业者和个人的会员制培训和参考咨询平台。

我们向会员开放成千上万本图书以及培训视频、学习路线、交互式教程和专业视频。这些资源来自 250 多家出版机构，其中包括 O’Reilly Media、Harvard Business Review、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Adobe、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett 和 Course Technology。

更多信息，请访问 http://oreilly.com/safari。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

我们为本书做了一个网页，把勘误信息、示例代码和其他附加信息列在了上面。地址是：

http://bit.ly/css-refactoring。

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：

http://facebook.com/oreilly

请关注我们的 Twitter 动态：

http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：

http://www.youtube.com/oreillymedia

致谢

写作本书的过程让我真切地感受到了自己的不足。

一天晚上，我心血来潮，毛遂自荐，给 O'Reilly Media 发了一封简单的邮件，附上了我对于这本书的构思。邮件发出去之后，我忐忑不安起来，但很快就把这事抛诸脑后了，因为我压根就没指望收到回信。

后来，我却收到了 O'Reilly 的回信，还是好消息。这令我更加不安起来，我怕自己被冲昏了头，但是跟 Simon St. Laurent、Brian MacDonald 和 Meg Foley 的合作非常愉快。我尤其感激本书的编辑 Meg Foley，因为她非常通情达理，不断地鼓励我，尽管我一次又一次没能在截止日期前交稿。我之前表示过歉意，现在容我再说一次——对不起，Meg！

本书中的技巧、策略和想法，是我多年来从各处收集来的。书中的大部分概念都不是我提出来的，因此，首先我想感谢前人贡献了这些概念。加入 Web 开发社区非常有好处，因为社区有很多聪明人，他们愿意跟别人分享自己的想法。我希望能通过此书为社区尽绵薄之力。

写一本书比我预想的难多了，写作过程很耗时，而且需要我独自完成，幸好工作时周围有好友相伴。每当写不下去时，我就想起 Andy Denmark，他的职业道德超好，我从他那里得到了不少鼓舞。Thor Denmark 还教我如何在顺境和逆境中都保持积极的态度，使我不至于陷入困境无法自拔。Nate Racklyeft 和 Josh Hudner 阅读了本书的初稿，并给出了大量宝贵的反馈意见，极大提高了本书的质量。Erin Wallace 从事设计工作，整天跟组织、过程和细节打交道，她的反馈意见也非常有价值，有助于我润色书稿，使其更容易理解。他们每天都关注着我，促使我每天都比前一天进步一点。为此，我对他们感激不尽。

此外，我还要郑重感谢 Christopher Schmitt，他审阅了本书。得知他审阅我的作品时，我觉得他的名字很熟悉，果不其然，我的案头上就摆着他的两本大作。Christopher，你的批注对我帮助很大。我刚开始学 CSS 时，你的著作让我受益匪浅。非常感谢你抽出时间帮助一个陌生人。希望我有机会报答你！

如果不感谢培养我、影响我、使我变成今天这个模样的家人，致谢部分就不完整。我的父母总是鼓励我读书，我现在仍努力多读书、少看电视。小时候，父亲就送给了我一本 O'Reilly 图书（我的第一本 O'Reilly 图书，我记得是讲 C 语言的）。我选择 IT 作为职业，很大程度上源于父亲的鼓励和他从事的职业。母亲和哥哥也是我的灵感源泉，并给予我巨大鼓舞，你们对我的关爱难以言谢。

最后，我想感谢咖啡。我爱你，咖啡。

电子书

扫描如下二维码，即可购买本书电子版。

第 1 章

重构和架构

本章是 CSS 重构之旅的起点，将介绍重构是什么，以及它与软件架构之间有什么关系。我们还将讨论重构的重要性以及你的代码也许需要重构的原因。我们将通过两个重构的示例，帮助你理解这些概念。

1.1　什么是重构

重构是指在不改变代码行为的前提下，重写代码，使其更加简洁、易于复用。如果你正在写代码，那么重构是你应该掌握的一项核心技能，因为不管你想不想，有时你都不得不重构代码。你也许已重构过代码，只是自己没有意识到。既然重构不改变代码的行为，那么学习重构之前你想先弄清楚重构的必要性也是可以理解的。但是，在回答这个问题之前，需要先来理解软件架构。

1.2　什么是软件架构

就像生物一样，软件系统通常由很多较小的部件组成，每个部件擅长做一件事。将这些部件组合起来，一起工作，可形成更大的软件系统。术语**软件架构**用来描述软件项目的各个不同部件之间的组合方式。

从简单的网站到复杂的宇宙飞船控制系统，每一种软件都有自己的架构，不管开发人员有意还是无意为之。然而，最好的架构通常在编码工作开始之前做过缜密规划。下面是优秀架构所具备的一些最重要的特点。

1.2.1 优秀架构是可预测的

软件架构**可预测**是指可以对软件的工作方式和结构做出准确的假设。可预测性表明预先的规划是合理的，并有助于节省开发时间，因为可以避免下列问题：

- 组件的功能是什么
- 某一段代码在何处
- 新代码加到何处

在可预测的架构中，人们可以做出精确的假设，不熟悉代码的开发人员也能够更快地理解该架构。

1.2.2 优秀架构可提升代码复用性

代码复用是指在多处使用同一代码而无需重写。代码复用优势明显，不用重写已有代码，可以加快开发速度。同理，解决某一问题所需的代码越少，维护所有用该代码实现的功能所需的时间就越少。例如，你在一段代码中发现了一处 bug，而由于项目多处用到该代码，故将 bug 带到了多处。你只需在一个位置修复该 bug，就可以相应修复其他各处的 bug。

1.2.3 优秀架构可扩展

可扩展性是优秀架构所遵循的一项原则，在具备该特点的系统上增加新功能很容易。大多数软件无法在一天之内开发完成，因此软件架构适于增量开发，且不需要做大的结构性变化，这一点非常重要。如果项目开发过程需要频繁地对架构做出较大的改动，发布将非常困难。

1.2.4 优秀架构可维护

跟可扩展性非常类似，**可维护性**对于架构也很重要。对于可维护的优秀架构，修改其现有功能很容易。随着时间的推进，需求也许会发生变化。迫于需求变动的压力，你将修改代码。可维护性软件是指你修改一处代码时，没必要大规模改动其他代码。

1.2.5 软件架构和重构

概括来讲，重构有助于维护和提升软件架构。重构就是指调整代码结构、使其更具意义的一套技术。重构可使代码可预测、可复用、可扩展和可维护。当你的软件架构具备了上述特点时，它对目标用户而言将更可靠，你在其上继续开发也会更加愉快！

1.3 需要重构的原因

为什么当初不把代码写正确，这样日后不就没必要重构了？尽管我们一心想设计和编写最

高质量的代码，但是随着时间的推移，**一些**因素将发生变化，导致需要重构代码。我们一起来看看其中几个原因。

1.3.1　需求变更

软件系统随着需求的变动而进化。软件在编写时是为了满足一组需求，可能没有考虑另一组需求，（没有写出来，也不应该写出来）。因此，需求变更时，代码也必须随之改变。此外，如果软件开发还有时间要求，有时会为优先满足功能的实现而走“捷径”，进而可能会影响代码质量。

1.3.2　架构设计不合理

即使你知道什么是优秀架构，投入大量时间事先规划好一切并不总是可行的。开发之初，你若不知道各组件的组合方式，开发过程也许需要重构。非常常见的做法是，快速开发一个新功能（可能会为实现功能而走“捷径”），以验证它对用户是否有吸引力。如确实能够吸引用户，再将代码整理干净；如达不到预期效果，则删除代码。

1.3.3　低估困难

预估软件开发需要多长时间很难，但不幸的是，我们常常根据估计结果来安排开发计划。如果项目开发周期被低估，将迫使开发人员“为了完成而完成”，导致他们快速编写代码，而不会花很多时间思考。如果该情况经常发生，即使最完美的代码也可能会变为一大盘子“意大利面条式代码”，难以理解和管理。

1.3.4　忽视最佳实践

跟上每一种最佳实践的发展步伐很难，当你的工作涉及多种技术和人员管理时更是如此。如果你们是一个团队协同工作，而你忽视了最佳实践，我希望有同事能提醒你。如果错过使用最佳实践的机会，日后你也许需要重新审查你的代码并进行一定程度的重构。

紧跟最佳实践的难点所在

技术发展日新月异，因此之前的最佳实践也许优势不再。例如，2011 年之前，在网站上展示具有圆角的容器，需要为每个角挂一张图片，将图片嵌入到 HTML 之中，然后用 CSS 为图片定位，确保各元素排列有序。如今这项技术已过时，因为现在的浏览器用 CSS 属性 `border-radius` 就能实现圆角。如果你不持续更新代码来使用这些最佳实践，你的技术债务将随时间的发展而增加，代码将变得非常糟糕。

1.4　什么情况下应该重构代码

结合代码的上下文重构代码会更加容易。因此，如果你修复的 bug 或开发的新功能用到了已有代码，重构是最好的选择。处理小任务时顺便重构代码，不至于把一切搞乱，他人若修改你重构过的代码也能从中受益。不断坚持重构代码，代码质量将达到卓越，前提是你的改动符合优秀架构的特点。

然而，有时你会遇到一段有很多依赖的代码，也许需要决定是否对其重构。重构有很多依赖的代码，就像抽衣服上的线：抽得越多，散开得越多。类似地，对于具有很多依赖的代码，你改动得越多，需要更新的依赖就越多。遇到这种情况，如果时间很紧，先把工作完成也许更适合，然后再匀出些时间，回头审视并重构代码。然而，如果开发过程中重构某些小功能不至于严重影响到开发计划，你也许可以考虑及时重构它们。

1.5　什么情况下不应该重构代码

知道什么情况下不应该重构，甚至比知道什么情况下应该重构更为重要。重构名声不太好，因为有时看起来软件开发人员只是为了重写而重写。代码也许是别人写的，没必要重构，但患有“不是我写的症”的开发人员一定要对其重构，因为他们认为不是自己写的代码就不是好代码。或者，有一天开发人员心血来潮，不再喜欢之前的代码编写方法（也许之前类名使用下划线而不是连字符，现在想改过来），因此他们钻入重构的兔子洞以求止痒。很多情况下，这些工作被视为“磨洋工”，它让人们感觉效率很高，事实上并非如此。第 5 章将讨论如何通过编写一套编码规范形成编码计划。那时，你将更加清楚，仅当重构能够改善架构或使代码符合编码规范时，才应进行重构。

1.6　我能重构自己的代码吗

如果你正在开发个人项目，答案为响亮的“能！”。但是如果你为组织工作，且不处在管理岗位，答案也许没有那么肯定。理想情况下，每个组织都理解重构的重要性，但现实往往并非如此。如果同事缺乏重构方面的技术知识，你也许可以尝试教教他们，别忘了推荐我这本书！

对软件项目的代码质量负责的聪明人很可能理解重构的意义，但是不能理解的人可能会持有以下意见：

- 花时间重写代码，却又看不到功能上的变化，既浪费时间，又浪费钱；
- 如果代码还能正常工作，没必要修复；
- 你应该当初就把代码写正确。

如果别人持有以上理由，而你对重构有足够的信心，我建议你重构代码，只要你能够保证

开发进度，并且小心谨慎，不破坏其他功能。如果你听到这些理由，我敢打赌他们从未参加过代码评审会议，因此你的改动可能不会被注意到。然而，如果只是为重构而重构，你也许需要考虑等到有必要改动时再重构；不成熟的优化往往跟技术债务同样糟糕。

1.7 重构示例

你对重构的好处和时机（以及何时不能重构）有了整体理解后，我们可以开始讨论怎样重构代码了。

尽管本书是讲代码重构的，但我们首先通过一段计算电子商务订单总价的代码和改变 HTML 元素样式的代码分析这个概念，这样理解起来会更容易。因此，我们的第一个示例将展示重构基本的 JavaScript 代码的方法，该代码计算电子商务订单的总价。 第二个示例将重构一部分 CSS 代码。

代码示例

长篇累牍的代码，或长达几页，或散布于多个文件，难以理解，故本书代码示例将使用简短的代码片段。第一个示例的所有 JavaScript 代码可以嵌入到 HTML 文件中，方便运行。

对于更复杂的示例，定义元素整体外观和样式的 CSS 将置于单独的 CSS 文件中。

本书代码中用到的行内样式（位于 `<style>` 和 `</style>` 标签之间）直接服务于当前示例，用来解释一个单独的概念。

本书所有代码均可从配套网站下载：https://www.cssrefactoringbook.com。

1.7.1 重构示例1：计算电子商务订单的总价

例 1-1 包含一段 JavaScript 代码，用户提供以下内容后，可计算电子订单的总价：

- 所购商品的单价
- 每种商品的购买数量
- 每种商品的单位运费
- 顾客的地址信息
- 可选择使用的、能降低订单价格的折扣码

例 1-1　计算电子商务订单总价

```
/**
 * 打过折、加入运费和税费之后，计算订单总价。
 *
```

```
 * @param {Object} customer——顾客信息，关于下订单者的一组信息。
 *
 * @param {Array.<Object>} lineItems——数组，包括所购商品、商品数量及每种商品的
                                    单位运费。
 *
 * @param {string} discountCode——可选择使用的折扣码，加入运费和税费之前使用该码。
 */
var getOrderTotal = function (customer, lineItems, discountCode) {
  var discountTotal = 0;
  var lineItemTotal = 0;
  var shippingTotal = 0;
  var taxTotal = 0;

  for (var i = 0; i < lineItems.length; i++) {
      var lineItem = lineItems[i];
      lineItemTotal += lineItem.price * lineItem.quantity;
      shippingTotal += lineItem.shippingPrice * lineItem.quantity;
  }

  if (discountCode === '20PERCENT') {
      discountTotal = lineItemTotal * 0.2;
  }

  if (customer.shiptoState === 'CA') {
    taxTotal = (lineItemTotal - discountTotal) * 0.08;
  }

  var total = (
      lineItemTotal -
      discountTotal +
      shippingTotal +
      taxTotal
  );

  return total;
};
```

使用例 1-2 中的数据，调用 `getOrderTotal` 函数，得到总价。程序输出 `Total: $266`。例 1-3 解释了为什么会输出这个结果。

例 1-2　用测试数据运行 `getOrderTotal` 函数

```
var lineItem1 = {
  price: 50,
  quantity: 1,
  shippingPrice: 10
};

var lineItem2 = {
  price: 100,
  quantity: 2,
  shippingPrice: 20
};
```

```
var lineItems = [lineItem1, lineItem2];

var customer = {
  shiptoState: 'CA'
};

var discountCode = '20PERCENT';

var total = getOrderTotal(customer, lineItems, discountCode);

document.writeln('Total: $' + total);
```

例 1-3　解释为什么 getOrderTotal 函数输出 Total: $266

```
discountTotal = 0
lineItemTotal = 0
shippingTotal = 0
taxTotal = 0

# FOR循环第1次迭代:
lineItemTotal = 0 + (50 * 1) = 50
shippingTotal = 0 + (10 * 1) = 10

# FOR循环第2次迭代:
lineItemTotal = 50 + (100 * 2) = 250
shippingTotal = 10 + (20 * 2) = 50

# discountCode为“20%”，计算discountTotal:
discountTotal = 250 * 0.2 = 50

# customer.shiptoState为“CA”，计算 taxTotal:
taxTotal = (250 - 50) * 0.08 = 16

total = 250 - 50 + 50 + 16 = 266
```

1. 单元测试

运行完计算过程，得到计算结果，一切似乎按照预期进行。为了确保每次都能得到正确结果，现在来编写单元测试。简单来讲，**单元测试**是指执行另一段代码的一段代码，以保证代码按照预期工作。单元测试应该用来测试单一功能，以缩小任何可能发现的问题的根本原因的范围。另外，发布任何新代码之前，都应该为你的项目编写一组单元测试并运行，以便尽早发现和修复引入系统的新 bug。

例 1-2 的输入数据可用来编写单元测试（见例 1-4），我们断言函数返回预期值（266）。运行完测试代码，将输出测试成功和失败的次数，对于没有通过测试的，将输出期望值和实际值。

例 1-4　为 getOrderTotal 函数编写的单元测试

```
var successfulTestCount = 0;
var unsuccessfulTestCount = 0;
var unsuccessfulTestSummaries = [];
```

```
/**
 * 断言getOrdertotal()函数计算正确。
 */
var testGetOrderTotal = function () {

    // 设定期望得到的结果

    var expectedTotal = 266;

    // 设定测试数据

    var lineItem1 = {
        price: 50,
        quantity: 1,
        shippingPrice: 10
    };

    var lineItem2 = {
        price: 100,
        quantity: 2,
        shippingPrice: 20
    };

    var lineItems = [lineItem1, lineItem2];

    var customer = {
        shiptoState: 'CA'
    };

    var discountCode = '20PERCENT';

    var total = getOrderTotal(customer, lineItems, discountCode);

    // 比较函数的计算结果与期望得到的结果

    if (total === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetOrderTotal: expected ' + expectedTotal + '; actual ' + total
      );
    }
};

// 运行测试

testGetOrderTotal();
document.writeln(successfulTestCount + ' successful test(s)<br/>');
document.writeln(unsuccessfulTestCount + ' unsuccessful test(s)<br/>');

if (unsuccessfulTestCount) {
    document.writeln('<ul>');
    for(var i = 0; i < unsuccessfulTestSummaries.length; i++) {
        document.writeln('<li>' + unsuccessfulTestSummaries[i] + '</li>');
```

```
        }
        document.writeln('</ul>');
    }
```

执行 testGetOrderTotal 函数，断言成功通过，如图 1-1 所示。

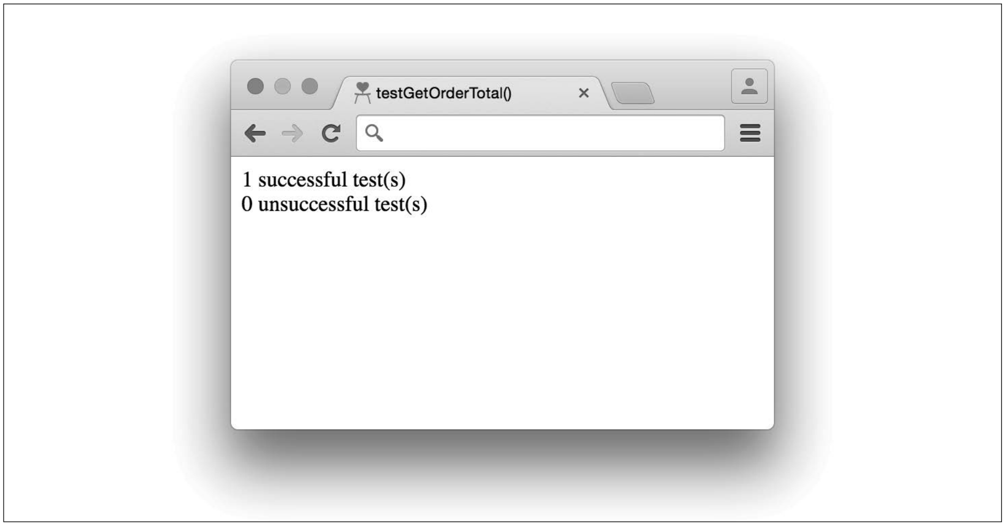


图 1-1：成功通过单元测试

然而，如果以后出于某种原因引入了 bug，导致计算 discountTotal 所用的乘数从 0.2 变为 −0.2，单元测试结果就会发生变化，我们将会得到图 1-2 所示的结果。



图 1-2：失败的单元测试

单元测试非常强大，可以保证你的系统按照预期工作。重写代码时，它的帮助尤其大，因为使用断言可帮助你确认代码的行为没有发生变化。

现在你理解了计算电子商务订单总价的代码，并实现了相应的单元测试代码，下面一起看看对其重构能带来哪些好处。

2. 重构getOrderTotal

仔细分析 `getOrderTotal` 函数，不难发现该函数内实现了多种计算：

- 从总价中减去的折扣
- 订单中所有商品的总价
- 总运费
- 总税额
- 订单总价

如果上述五项中任意一项的计算过程引入 bug，单元测试（`testGetOrderTotal`）将告诉我们出错了，但是不会**明确**指出 bug 的位置。这正是单元测试应该测试单一功能的主要原因。

为了让代码所实现功能的粒度更细，上面提到的这些计算都应该抽取出来，作为单独的一个函数，并且用能够描述其功能的名称作为函数名，请见例 1-5。

例 1-5　抽取代码片段，形成新函数

```
/**
 * 计算所有line items的总价。
 *
 * @param {Array.<Object>} lineItems——数组，包括所购商品、商品数量及
                                      每种商品的单位运费。
 *
 * @returns {number}——所有line items的总价。
 */
var getLineItemTotal = function (lineItems) {
    var lineItemTotal = 0;

    for (var i = 0; i < lineItems.length; i++) {
         var lineItem = lineItems[i];
         lineItemTotal += lineItem.price * lineItem.quantity;
    }

    return lineItemTotal;
};

/**
 * 计算所有line items的总运费。
 *
 * @param {Array.<Object>} lineItems——数组，包括所购商品、商品数量及
                                      每种商品的单位运费。
 *
 * @returns {number}——所有line items的运费。
```

```
*/
var getShippingTotal = function (lineItems) {
    var shippingTotal = 0;

    for (var i = 0; i < lineItems.length; i++) {
        var lineItem = lineItems[i];
        shippingTotal += lineItem.shippingPrice * lineItem.quantity;
    }

    return shippingTotal;
};

/**
 * 计算一个订单的总价按照折扣减去了多少钱。
 *
 * @param {number} lineItemTotal——所有line items的总价。
 *
 * @param {string} discountCode——可选择使用的折扣码，加入运费和税费之前使用该码。
 *
 * @returns {number}——订单总价按照折扣减去了多少钱。
 */
var getDiscountTotal = function (lineItemTotal, discountCode) {
    var discountTotal = 0;

    if (discountCode === '20PERCENT') {
        discountTotal = lineItemTotal * 0.2;
    }

    return discountTotal;
};

/**
 * 计算一个订单应缴纳的总税费。
 *
 * @param {number} lineItemTotal——所有line items的总价。
 *
 * @param {Object} customer——顾客信息，关于下订单者的一组信息。
 *
 * @returns {number}——一个订单应缴纳的总税费。
 */
var getTaxTotal = function () {
    var taxTotal = 0;

    if (customer.shiptoState === 'CA') {
        taxTotal = lineItemTotal * 0.08;
    }

    return taxTotal;
};
```

我们应该为每个新函数编写如例 1-6 所示的单元测试。

例 1-6　用 JavaScript 为新抽取出来的函数编写的单元测试

```
/**
 * 断言getLineItemTotal的计算结果符合预期。
```

```
 */
var testGetLineItemTotal = function () {
    var lineItem1 = {
        price: 50,
        quantity: 1
    };

    var lineItem2 = {
        price: 100,
        quantity: 2
    };

    var lineItemTotal = getLineItemTotal([lineItem1, lineItem2]);
    var expectedTotal = 250;

    if (lineItemTotal === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetLineItemTotal: expected ' + expectedTotal + '; actual ' +
          lineItemTotal
      );
    }
};

/**
 * 断言getShippingTotal的计算结果符合预期。
 */
var testGetShippingTotal = function () {
    var lineItem1 = {
        quantity: 1,
        shippingPrice: 10
    };

    var lineItem2 = {
        quantity: 2,
        shippingPrice: 20
    };

    var shippingTotal = getShippingTotal([lineItem1, lineItem2]);
    var expectedTotal = 250;

    if (shippingTotal === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetShippingTotal: expected ' + expectedTotal + '; actual ' +
          shippingTotal
      );
    }
};

/**
```

```
 * 确保使用有效的折扣码时，GetDiscountTotal的计算结果符合预期。
 */
var testGetDiscountTotalWithValidDiscountCode = function () {
    var discountTotal = getDiscountTotal(100, '20PERCENT');
    var expectedTotal = 20;

    if (discountTotal === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetDiscountTotalWithValidDiscountCode: expected ' + expectedTotal +
          '; actual ' + discountTotal
      );
    }
};

/**
 * 确保使用无效的折扣码时，GetDiscountTotal的计算结果符合预期。
 */
var testGetDiscountTotalWithInvalidDiscountCode = function () {
    var discountTotal = get_discount_total(100, '90PERCENT');
    var expectedTotal = 0;

    if (discountTotal === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetDiscountTotalWithInvalidDiscountCode: expected ' + expectedTotal +
          '; actual ' + discountTotal
      );
    }
};

/**
 * 确保顾客住在加利福尼亚时，GetTaxTotal的计算结果符合预期。
 */
var testGetTaxTotalForCaliforniaResident = function () {
    var customer = {
      shiptoState: 'CA'
    };

    var taxTotal = getTaxTotal(100, customer);
    var expectedTotal = 8;

    if (taxTotal === expectedTotal) {
      successfulTestCount++;
    } else {
      unsuccessfulTestCount++;
      unsuccessfulTestSummaries.push(
          'testGetTaxTotalForCaliforniaResident: expected ' + expectedTotal +
          '; actual ' + taxTotal
      );
```

```
        }
    };

    /**
     *确保顾客不住在加利福尼亚时，GetTaxTotal的计算结果符合预期。
     */
    var testGetTaxTotalForNonCaliforniaResident = function () {
        var customer = {
            shiptoState: 'MA'
        };

        var taxTotal = getTaxTotal(100, customer);
        var expectedTotal = 0;

        if (taxTotal === expectedTotal) {
          successfulTestCount++;
        } else {
          unsuccessfulTestCount++;
          unsuccessfulTestSummaries.push(
              'testGetTaxTotalForNonCaliforniaResident: expected ' + expectedTotal +
              '; actual ' + taxTotal
          );
        }
    };
```

最后，我们用这些新抽取出来的函数改写 `getOrderTotal` 函数，请见例 1-7。

例 1-7　用新抽取出来的函数改写 `getOrderTotal` 函数

```
    /**
     *打过折、加入运费和税费之后，计算订单总价。
     *
     * @param {Object} customer——顾客信息，关于下订单者的一组信息。
     *
     * @param {Array.<Object>} lineItems——数组，包括所购商品、商品数量及
                                          每种商品的单位运费。
     *
     * @param {string} discountCode——可选择使用的折扣码，加入运费和税费之前使用该码。
     */
    var getOrderTotal = function (customer, lineItems, discountCode) {
        var lineItemTotal = getLineItemTotal(lineItems);
        var shippingTotal = getShippingTotal(lineItems);
        var discountTotal = getDiscountTotal(lineItemTotal, discountCode);
        var taxTotal = getTaxTotal(lineTtemTotal, customer);

        return lineItemTotal - discountTotal + shippingTotal + taxTotal;
    };
```

分析完上述代码，我们观察到：

- 函数比以前更多了；
- 单元测试比以前更多了；
- 每个函数实现一个特定功能；

- 每个函数都有一个单元测试；
- 多个函数组合起来可以实现更复杂的计算。

总体来讲，重构之后代码结构更加合理。`getOrderTotal` 函数内部计算各种价格的代码被抽取出来，作为一个个单独的函数，而且每个函数都有相应的单元测试。这意味着当代码中引入 bug 时，更容易定位受影响的功能。此外，如果总税额或运费需要更换计算方式，而现有功能已经提供了可用的单元测试，那么更换之后，可方便地用单元测试加以验证。

1.7.2　重构示例2：重构CSS的简单示例

例 1-8 的代码用来展示网站的标题栏。

例 1-8　网站标题栏代码

```
<!doctype html>
<html>
  <head>
    <title>Ferguson's Cat Shelter</title>
    <link rel="stylesheet" type="text/css" href="css/style.css" />
  </head>
  <body>
    <main>
      <h1 style="font-family: Helvetica, Arial, sans-serif;font-size: 36px;
        font-weight: 400;text-align: center;">
        San Francisco's Premiere Cat Shelter
      </h1>
    </main>
  </body>
</html>
```

打开浏览器，加载 index.html，将看到如图 1-3 所示的内容。

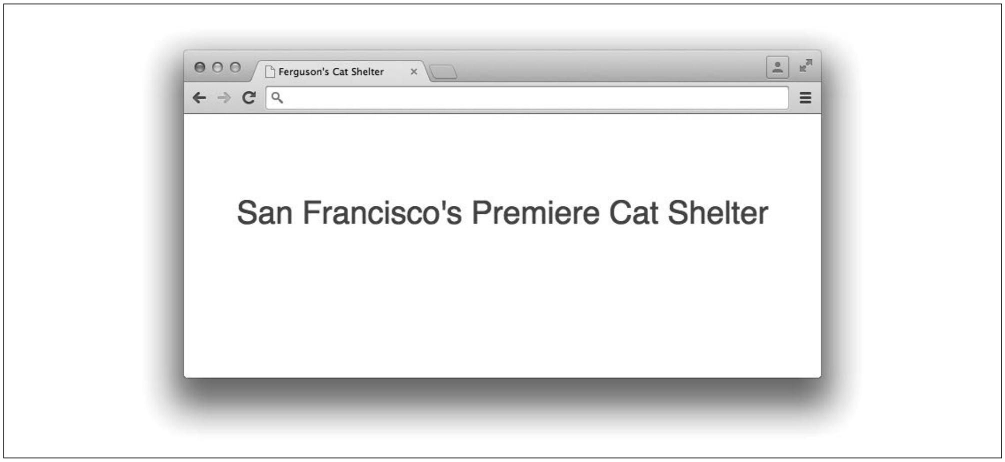


图 1-3：网站标题栏截图

在第一个重构示例中，重构之前我们编写了单元测试，以保证重构没有改变代码的行为。重构 CSS 时，确保重写代码没有改变展示效果同样重要。但无法直接测试，因为重构 CSS 带来的是视觉上的变化，而不是明确的数值上的变化。第 5 章将讨论保持视觉效果上的对等性所用到的一些技术。现在，在重构前截图作为参考即可。

重构网站标题

显然，例 1-8 的代码还有提升空间，因为用 `<h1>` 标签表示的标题栏样式内嵌在 `style` 属性之中。通过元素的 `style` 属性或将样式置于 `<style></style>` 标签之间，将样式嵌入到 HTML 代码之中，这种样式叫作**行内样式**。

行内样式复用性不高，这一点非常类似于例 1-1 重构前、函数体内执行多种计算的函数。使用 `style` 属性设置的样式，只可用于当前元素。嵌入到 `<style></style>` 标签中的样式，只可用于当前页。

因为大多数网站包含多个页面，并且每个页面都可能有一个标题栏，所以标题的样式应该抽取出来作为单独的 CSS 文件（该例中的 style.css），以用于多个网页，并被浏览器缓存。style.css 文件的内容请见例 1-9，例 1-10 为抽掉行内 CSS 的 HTML 代码。

例 1-9　将标题栏 CSS 抽取出来作为 style.css 文件

```
h1 {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 36px;
  font-weight: 400;
  text-align: center;
}
```

例 1-10　抽掉行内 CSS 的 HTML 代码

```
<!doctype html>
<html>
  <head>
    <title>Ferguson's Cat Shelter</title>
    <link rel="stylesheet" type="text/css" href="css/style.css" />
  </head>
  <body>
    <main>
      <h1>San Francisco's Premiere Cat Shelter</h1>
    </main>
  </body>
</html>
```

刷新浏览器，很快就会发现页面显示效果没有变化，我们由此得出以下结论：

- 抽取行内 CSS 可提升复用性；
- 分离代码功能（样式和结构）可增强代码可读性；
- 回归测试可手动用 Web 浏览器完成，或通过比较重构后的界面和重构前的截图完成。

将样式抽取出来作为单独的文件，可提升代码的复用性，因为抽取出来的样式可用于多个文件。将 CSS 置于一个独立于 HTML 代码的文件后，HTML 和 CSS 都更加清晰易读，因为 HTML 中不会包含冗长的样式定义，并且 CSS 可以按照符合逻辑的方式以声明块的形式组织在一起。最后，测试重构是否改变了页面；可手动用浏览器重新加载页面，与重构前的截图进行对比。

虽然该示例很简单，但大量类似的小改动累积起来，效果将十分可观。

1.8 总结

通过本章的学习，我们知道了什么是重构以及它与软件架构之间的关系。我们还了解了重构为什么重要，以及何时应该重构。最后，通过两个重构的例子了解了单元测试。下一章介绍级联。要编写 CSS，级联是需要理解的最重要的概念。

第 2 章

级联

可以说没有比编写了一大堆 CSS，结果发现生效的不是目标样式而是其他样式更糟糕的了。为了更好地理解发生这种现象的原因，本章将解释 Web 浏览器如何使用级联决定为元素应用某种样式。

2.1　什么是级联

级联是浏览器决定为元素应用哪种样式的一种方法。因为同一元素可以应用多种样式，所以遇到元素所用样式与预期不符时，理解级联的工作原理很重要。好在，级联并没有像它听起来那么复杂；样式根据选择器的特指度以及规则集出现的次序起作用。

2.2　选择器特指度

特指度度量的是 CSS 选择器识别元素的精确性。为选取元素，不同类型的选择器常组合在一起，计算特指度时需要分析这些选择器（除了通用选择器 *）。为 (`a`，`b`，`c`，`d`) 中的各个变量赋予相应的数值，就能得到特指度。

(1) 如果用 `style` 属性应用样式，则 `a`=1，否则 `a`=0。
(2) `b` 为 ID 选择器的数量。
(3) `c` 为类选择器、属性选择器和伪类的数量。
(4) `d` 为类型选择器和伪元素的数量。

以上各量计算完成后，这些量拼接起来可得到特指度。为具体理解，请思考例 2-1 中的选择器。

例 2-1　计算规则集的特指度

```
#nav-global > ul > li > a.nav-link {
    color: #000000;
}
```

用刚刚定义的算法，我们可以确定该选择器的特指度为 (0, 1, 1, 3)：

(1) 样式不是用 `style` 属性添加的，因此 a=0
(2) 只有 1 个 ID 选择器（`#nav-global`），因此 b=1
(3) 只有 1 个类选择器（`.nav-link`），因此 c=1
(4) 有 3 个类型选择器（`ul`、`li` 和 `a`），因此 d=3

比较选择器的特指度时，最左侧的选择器特指度最高。如果最左侧变量的两个值相等，需要比较右侧紧挨着的变量的值。举例来说，特指度 (1, 0, 0, 0) 高于 (0, 1, 1, 3)，同理，(0, 2, 1, 3) 高于 (0, 1 ,1, 3)。然而，特指度 (0, 1, 1, 3) 低于 (0, 1, 1, 4) 或 (0, 1, 2, 0)。更多计算特指度的例子请见例 2-2。

例 2-2　特指度计算示例

```
/**
 * 该选择器特指度为(0,0,2,2)，因为它有：
 *   0个行内样式
 *   0个ID
 *   1个类选择器（.title）、0个属性选择器和1个伪类选择器（:first-child）
 *   2个类型选择器（li和h2）
 */
li:first-child h2 .title {}

/**
 * 该选择器特指度为(0,1,2,1)，因为它有：
 *   0个行内样式
 *   1个ID（#nav）
 *   1个类选择器（.selected）、0个属性选择器和1个伪类选择器（:hover）
 *   1个类型选择器（a）
 */
#nav .selected > a:hover {}

/**
 * 该选择器特指度为(0,1,2,3)，因为它有：
 *   0个行内样式
 *   1个ID（#nav）
 *   1个类选择器（.selected）、0个属性选择器和1个伪类选择器（:hover）
 *   3个类型选择器（html、body和a）
 */
html body #nav .selected > a:hover {}
```

2.3 规则集顺序

规则集顺序描述的是一个 CSS 规则集在样式表中的位置。如果两个声明块中的选择器特指度相同，且它们为同一元素的某个属性应用样式，那么在样式表中处于相对靠后位置的声明块中的属性优先级较高。这表明例 2-3 中的 `color` 属性使用值 `#000000`，因为它应用的是样式表后面、特指度跟前面相等的声明块中的颜色样式。

例 2-3 用后面声明块定义的色值来为 color 属性赋值

```
<!doctype html>
<html>
  <head>
    <title>Inline Styles and Specificity</title>
    <style type="text/css">
      #nav-global > ul > li > a.nav-link {
        color: #FFFFFF;
      }
      #nav-global > ul > li > a.nav-link {
        color: #000000;
      }
    </style>
  </head>
  <body>
    <nav id="nav-global">
      <ul>
        <li>
          <a href="#" class="nav-link">Link</a>
        </li>
      </ul>
    </nav>
  </body>
</html>
```

2.4 行内CSS和特指度

除非用 `style` 属性添加行内样式，否则特指度和规则集顺序决定为元素应用什么样式。例 2-4 演示了为锚点标签添加行内样式。因为在行内样式中为该锚点元素设置了 `color` 属性，所以将使用该属性值。不管 `<style>` 块或外部样式表中的选择器有多么精确，它们都比不上为元素添加的行内样式精确。

例 2-4 通过 `style` 属性添加行内 CSS 样式来为 `color` 属性赋值

```
<!doctype html>
<html>
  <head>
    <title>Inline Styles and Specificity</title>
    <style type="text/css">
      #nav-global > ul > li > a.nav-link {
        color: #000000;
      }
```

```
    </style>
  </head>
  <body>
    <nav id="nav-global">
      <ul>
        <li>
          <a href="/" class="nav-link" style="color: #1200FF;">Link</a>
        </li>
      </ul>
    </nav>
  </body>
</html>
```

2.5　用!important声明覆盖级联样式

<style> 块或外部样式表中的样式，如要比其他样式（包括用 style 属性添加的行内样式）更精确，唯一的方法是在声明块中添加 !important。这要求浏览器凡是遇到与给定规则集选择器相匹配的元素，都应用该声明块中定义的样式，而不管具有较高特指度的选择器所定义的属性值。若选择相同元素的多个声明块均使用 !important，则位置最靠后的声明块起作用。

例如，在例 2-5 中，锚点标签的文本为白色（#FFFFFF），因为第一条规则集使用了 !important 声明。如果两条规则集都添加了 !important，那么锚点标签的文本为黑色（#000000），因为相对靠后的规则集起作用。如果没有使用 !important，锚点标签的文本为蓝色（#1200FF），因为行内样式特指度最高。注意，!important 不能添加到行内样式 style 属性之中（即 Link 这种写法是错误的）。

例 2-5　在位置相对靠前的声明块中添加 !important 来为 color 属性赋值

```
<!doctype html>
<html>
  <head>
    <title>Inline Styles and Specificity</title>
    <style type="text/css">
      #nav-global > ul > li > a.nav-link {
        color: #FFFFFF !important;
      }
      #nav-global > ul > li > a.nav-link {
        color: #000000;
      }
    </style>
  </head>
  <body>
    <nav id="nav-global">
      <ul>
        <li>
          <a href="/" class="nav-link" style="color: #1200FF;">Link</a>
        </li>
      </ul>
```

```
    </nav>
  </body>
</html>
```

2.6 总结

扎实地理解级联和特指度的计算方法后，再去学习更多与重构相关的知识将会更加容易，因为重构是以这些概念为基础的。在后续几章的学习过程中，一定要思考对于每个概念，级联在其中扮演什么角色。搞清楚这个问题之后，一切是怎么组合在一起的将会更加明朗。下一章，我们来了解如何编写更优秀的 CSS。

第 3 章

编写更优质的CSS

关于编写 CSS 的“最佳实践”是什么是颇有争议的。这是因为实现一种效果的方法有很多种，单纯说一种技术好于另一种可能非常主观。然而，我们在第 1 章中已经讲过，优秀的架构是可预测、可维护和可扩展的，且其能提升代码的可复用性。带着对这一定义的认识，我们来学习本章中的概念，以便帮你打下坚实的基础，编写出更优质的 CSS。

3.1 使用注释

注释记录了编码过程中的重要内容，在我们日后阅读代码时，它能起到辅助理解的作用。注释应该记录的内容包括：

- 文件内容
- 选择器的依赖、用法等
- 使用特定声明的原因（因为浏览器的怪癖而使用时，予以说明帮助尤其大）
- 正在被重构的、不应该继续使用的废弃样式

CSS 只有块级注释（注释语句可以折行），注释以 /* 开始，以 */ 结束。若注释只需要一行，仍旧可以用块级注释，但是同样必须以 /* 开始，以 */ 结束。下面来看几个注释的示例：

```
/*
 * 主导航链接的样式。
 *
 * @see templates/_navigation.html
 */
```

```
.nav-link {
  padding: 4px;
  text-decoration: none;
}

.nav-link:hover {
  border-bottom: 4px solid #000000;

/*
 * 防止因增加了4px下边框而导致元素移动
 */
padding-bottom: 0;
}

/* @deprecated[1] */
.navigation-link {
  color: #1200FF;
}
```

3.2 结构一致的规则集

规则集可以写到一行，也可以分作多行。例 3-1 所示的规则集只有一行，例 3-2 则分为多行。编写规则集时，花括号甚至可以自成一行，如例 3-3 所示。

例 3-1　只占一行的 CSS 规则集

```
selector { property1: value; property2: value; property3: value; }
```

例 3-2　跨越多行的 CSS 规则集

```
selector {
    property1: value;
    property2: value;
    property3: value;
}
```

例 3-3　跨越多行的 CSS 规则集，花括号自成一行

```
selector
{
    property1: value;
    property2: value;
    property3: value;
}
```

编写规则集时，注意保持格式的一致性，以使 CSS 可预测性更强，更易于理解。

这当然与个人偏好有关，我喜欢将规则集的每个声明单独写到一行，如例 3-2 所示。此外，我还喜欢按照字母顺序排列属性，这样 CSS 属性可预测性再次加强，属性更易于查找。

注 1：“@deprecated”，表示之前曾用过该样式，但建议其他程序员以后不要再用。——译者注

用浏览器引擎前缀组织属性

浏览器引擎前缀是指浏览器厂商在新的实验性质的 CSS 属性实现标准化之前，预先在属性前添加的字符串。常见的前缀有以下几个：为使用 Blink 或 WebKit 渲染引擎的浏览器（Chrome 和 Safari）添加的 `-webkit-`；为使用 Gecko 渲染引擎的浏览器（Firefox）添加的 `-moz-`；为使用 Trident 渲染引擎的浏览器（Internet Explorer/Edge）添加的 `-ms-`。属性标准化之后，将不再为其添加前缀。

例如，`transform-origin` 就是一个这样的属性。你可以通过该属性修改元素发生变换（比如旋转和平移）的基点位置。现在我们要想使用该属性，应该用添加了浏览器引擎前缀的写法，把标准写法作为备用：

```
-ms-transform-origin: @origin;
-moz-transform-origin: @origin;
-webkit-transform-origin: @origin;
transform-origin: @origin;
```

这些属性的顺序非常重要，因为浏览器按照从上到下的顺序应用声明块的属性，它忽略无法识别的属性，应用能识别的属性。这表明将没有前缀的标准 CSS 属性放到添加有前缀的属性前面，可能导致在两者都支持的浏览器上前者被后者覆盖。

既不支持新属性，又不支持带有前缀的属性的老式浏览器，会把这两种属性都忽略掉。只支持带有前缀的新属性的老式浏览器，将采用恰当的属性，忽略不带前缀的标准写法。更新一点的浏览器，为了保持向后兼容，继续支持新属性带有前缀的写法，同时支持不带前缀的标准写法。对于这种浏览器，这两种属性都可以。最后，不再支持属性带前缀写法，仅支持不带前缀的标准写法的浏览器，将忽略带前缀的属性，应用标准属性。

因为某一特定浏览器的所有用户升级浏览器需要一定的时间，所以带有前缀和不带前缀的写法都应该保留到你的网站不再支持该浏览器为止。

功能标记

带前缀的 CSS 属性增加了维护成本，因为它们使得样式表急剧膨胀。为了弥补这一弱点，很多浏览器厂商转而使用选择性加入（opt-in）的功能，以便让开发人员实验最新的 CSS 属性。如果用户仍使用老式浏览器访问你的网站，你需要支持它们，然而，你也许还想继续支持带前缀的 CSS 属性。

3.3 保持选择器的简单

将多种不同的选择器和结合符写到一起形成的选择器非常复杂。然而，选择器能够做到非常精确，并不意味着它们就应该很精确，比如例 3-4。

例 3-4　用于选择某一特定元素的 CSS 选择器

```
<!doctype html>
<html>
  <head>
    <title>Keep Selectors Simple</title>
    <style type="text/css">
      div > nav > ul > li > a {
          color: #1200FF;
      }
    </style>
  </head>
  <body>
    <div>
      <nav>
        <ul>
          <li>
            <a href="./policies.html">Policies</a>
          </li>
        </ul>
      </nav>
    </div>
  </body>
</html>
```

例 3-4 的代码使用了多个子选择器（>）为一个特定锚点标签添加样式。用该方法为锚点标签元素添加样式不太好，因为所用的选择器高度依赖页面的 HTML 结构。一旦 HTML 结构改变，目标样式将不起作用。因此更佳的做法是为 HTML 元素增加一个类，并为该类增加样式，详见例 3-5。

例 3-5　增加类后的 HTML 结构

```
<!doctype html>
<html>
  <head>
    <title>Keep Selectors Simple</title>
    <style type="text/css">
      a.nav-link {
          color: #1200FF;
      }
    </style>
  </head>
  <body>
    <div>
      <nav>
        <ul>
          <li>
            <a href="./policies.html" class="nav-link">Policies</a>
          </li>
        </ul>
      </nav>
    </div>
  </body>
</html>
```

例 3-5 中所有复杂的子选择器均已删除，用类选择器选择类为 `nav-link` 的元素。然而，选择器 `a.nav-link` 仍然比实际需要的更加精确：我们将其称为**精确过头的选择器**，因为只能用其选择锚点标签。

如例 3-6 所示，该选择器还可以进一步简化。

例 3-6　简化过的选择器

```
.nav-link {
  color: #1200FF;
}
```

如上所示，尽可能地简化选择器有很多优势，因为样式不再依赖于页面的 HTML 结构，CSS 文件也会略微缩小。假如元素应该使用不同于例 3-6 中所用的样式，只要与父容器一起更换即可（例如：`.parent-container .nav-link{ color: #FF0000; }`），这不仅为元素应用了其他样式，还为该样式的使用场景提供了背景信息。

然而，有时使用精确的选择器**确实**合理——例如，为某一元素添加一个类，实现有别于将该类应用于其他元素时得到的样式。

在例 3-7 中，`error` 类为文本应用红色样式。为输入框增加 `error` 类样式时，浏览器也许会为文本和文本框应用红色样式，而不改动其他使用 `error` 类的元素的样式。`input.error` 的精确程度足以为输入框元素的文本和边框颜色增加样式，因此它被认为是与精确过头的选择器相对的**合格的选择器**。

例 3-7　可接受的合格选择器

```
.error {
  color: #FF0000;
}

input.error {
  border-color: #FF0000;
}
```

高性能选择器

简单的选择器比起复杂的选择器性能更高；然而，随着计算机性能的持续提升和浏览器的持续优化，大多数时候我们都不用过于担心选择器的性能。当然，还是应该优先选用简单的选择器，这是因为简单的选择器复用程度高，易于理解，而并不是因为它们更高效，虽然这一点显而易见。遵循本书的指南，你不必陷入对选择器性能的恐慌，但是对选择器性能有个整体认识仍然很有必要。

1. 从右向左匹配选择器

Web 浏览器需要快速选择并为元素应用样式，以便尽快为用户提供可用的网页。浏览器从右向左匹配选择器，因此它能够忽略前面不匹配的元素，而不必把时间浪费到确认元素

是否匹配上。我们结合例 3-8 来理解此概念，该段代码中包含类 `.nav-link` 和一个 `strong` 元素。

例 3-8　一段层级结构简单的 HTML 代码

```
<!doctype html>
<html>
  <head>
    <title>Another Example</title>
  </head>
  <body>
    <div>
      <nav>
        <ul>
          <li>
            <strong>Not a Link</strong>
          </li>
        </ul>
      </nav>
     </div>
     <div>
      <nav>
        <ul>
          <li>
            <a href="#" class="nav-link">Link</a>
          </li>
        </ul>
      </nav>
    </div>
  </body>
</html>
```

如果用 `div > nav > ul >li > a` 选择锚点标签，并且浏览器尝试从左向右匹配元素（排除 `<!doctype>`），它需要执行下列步骤。

(1) 遍历每个元素，检查它是否是 `<div>` 元素。
(2) 检查步骤 1 匹配到的每个 `<div>` 元素，确认它是否有子节点 `<nav>`。
(3) 检查步骤 2 匹配到的每个 `<nav>` 元素，确认它是否有子节点 `<ul>`。
(4) 检查步骤 3 匹配到的每个 `<ul>` 元素，确认它是否有子节点 `<li>`。
(5) 检查步骤 4 匹配到的每个 `<li>` 元素，确认它是否有子节点 `<a>`。
(6) 为找到的那一个 `<a>` 元素应用样式。

反过来讲，如果同一选择器（`div > nav > ul > li > a`）从右向左匹配，浏览器需要执行下列步骤。

(1) 遍历每个元素，检查它是否是 `<a>` 元素。
(2) 检查步骤 1 匹配到的每个 `<a>` 元素，确认它是否有一个 `<li>` 父节点。
(3) 检查步骤 2 匹配到的每个 `<li>` 元素，确认它是否有一个 `<ul>` 父节点。

(4) 检查步骤 3 匹配到的每个 `<ul>` 元素，确认它是否有一个 `<nav>` 父节点。
(5) 检查步骤 4 匹配到的每个 `<nav>` 元素，确认它是否有一个 `<div>` 父节点。
(6) 为与选择器相匹配的 `<a>` 元素应用样式。

虽然这两种顺序步骤数相同，但存在较大的区别：从左向右匹配时，两个 `<div>` 元素都需要匹配一遍。而从右向左匹配时，浏览器能够直接排除不包含锚点标签的整个 `<div>`。若要进一步提升性能，还可以用 `.nav-link` 类选择锚点标签，那么浏览器只需要遍历每个元素，检查它是否包含该类。

对浏览器如何解析选择器有了以上大致的理解之后，我们来分析一个更加极端（很容易避免）的例子。

2. 关键选择器

例 3-9 中的选择器，选择的是祖先元素为 `<body>` 标签的任意元素。从右向左解析该选择器，能够选择页面的每个元素，并向上回溯查看它是否有个 `<body>` 祖先元素。该选择器效率极低，因为几乎每个可见元素都是 `<body>` 元素的后代。

例 3-9　以 body 为祖先元素的通用选择器

```
body * {
  font-size: 12px;
}
```

正如前面讲过的，浏览器从右向左匹配元素，因此它能够及时排除与选择器不匹配的元素。选择器最右边的部分叫作**关键选择器**，例 3-9 是以通用选择器 `*` 作为关键选择器的。

单独使用通用选择器为所有元素（`* {}`）应用样式，浏览器可以很快完成渲染工作，因为它只需要匹配页面的每个元素。然而，当通用选择器与另一个选择器和结合符（例 3-9 中的祖先选择器）配合使用时，浏览器匹配合适的元素所做的工作要更多。只使用通用选择器，不要将其与结合符和其他选择器配合使用可以解决该问题。

3.4　分离CSS和JavaScript

由于 JavaScript 和 CSS 两者都依赖于 HTML 元素的类和 ID，因此它们可能会混杂在一起。此外，因为 JavaScript 能够修改 HTML 元素的样式，所以这两种语言的职责可能会变得很混乱。为了区分 CSS 和 JavaScript 的职能，JavaScript 中用来选择元素的类和 ID，不应该再用来为元素添加样式。类似地，用 JavaScript 修改元素样式时，应该通过增加和删除类来实现。

3.4.1　在JavaScript中使用带前缀的类和ID

HTML 中的类既用来为元素添加 CSS 样式，又在 JavaScript 用作选择器，这种情况很常

见。同样很常见的是，添加到元素中的类和 ID 只用于选择元素，而不是为元素添加样式。HTML 中的类和 ID，若未在 CSS 中使用，这将使得我们难以找到能够修改元素样式的 CSS。类似地，如果为了让类名更切合它所作用的元素而修改了类名，却没有同步修改 JavaScript 中的类名，JavaScript 代码将不起作用。

比较简单的修改方法是，在只用于 JavaScript 的类和 ID 前添加 `js-`。例如，如果我们要在 JavaScript 中选择与政策相关的一组选项卡，那么可以用 `js-tab-group-policies` 作为 ID。只用添加了 `js-` 前缀的类和 ID 作为 JavaScript 选择器，就可以消除 JavaScript 和 CSS 之间的依赖关系。

3.4.2　用类修改元素样式

元素样式可以用 JavaScript 修改，很多库（像 jQuery）简化了具体的实现方法。然而，用 JavaScript 修改样式通常表示用 `style` 属性为元素增加行内样式，这将使得行内样式最精确。此外，在 JavaScript 中修改样式，表示 JavaScript 要为特定的 CSS 样式负责，这超出了它的职责范围。如遇到元素样式需要改动的情况，则不仅要在 CSS 文件中查找现有样式，还要在 JavaScript 文件中查找。

因此，若要修改 HTML 元素样式，不要用 JavaScript 添加 `style` 属性，而是应该为元素增加或删除类。这样不仅可以应用合适的样式，该元素的 CSS 样式集也能跟其余的网站 CSS 合理地组织在一起。

3.5　使用类

用类和 ID 识别 DOM 中的元素，为其应用特定样式很方便。每个网页中，类可以根据需要复用多次，并且它们的特指度很低，因此可以方便地覆盖。而 ID 几乎正好与此相反，它们的特指度很高，因此无法轻易覆盖，并且同一个网页中每个 ID 最多只能用一次。对于持续变化的网站，在为其编写 CSS 时，应该用类为元素增加样式。

跟其他很多问题一样，对于是否只应该使用类，前端开发人员持有不同的意见。他们反对只使用类的一个原因是 ID 不仅合法，还有助于保证 HTML 结构体只被使用一次。例如，如果网站的一个页面要实现两栏的效果，其中一栏为侧边栏，另一栏放置页面主要内容，那么我们可以用下面的选择器：

```
#content {
/* # 内容区域的样式写到这里 */
}
#sidebar {
/* # 侧边栏的样式写到这里 */
}
```

ID 的使用表明这些元素最多只能在页面中用一次。然而，如果日后想修改网页，再增加一道侧边栏，将内容置于两道侧边栏之间，又该怎么做呢？第二道侧边栏就需要用新的 ID 或类为其添加新的 CSS，或将 `#sidebar` 改为类，重用其样式。

使用独一无二的 ID 更具优势的元素，通过使用单独的类也能实现相同的目标。如果日后该样式需要复用，不用修改直接用即可。从实际应用来看，大多数情况下，类和 ID 在 CSS 性能方面没有明显的差别。

> 在 JavaScript 中，选择元素的最快方式是 ID。不使用 ID 为元素增加样式，是将 CSS 和 JavaScript 分离的另外一个好方法，该方法非常类似于为类和 ID 增加 `js-` 前缀。

3.6 类名要有意义

有意义的类名通过描述为什么元素增加样式，从而提供恰如其分的背景信息。这样既可以防止因细节过少而表意模糊，又可以避免因信息过多而妨碍代码重用。

例 3-10 演示的是用类选择元素，但是 `a` 是什么？含义模糊的类名会令人费解。也许 `.a` 用来表示“animal”（动物）。若确实如此，用 `.animal` 则更贴切，因为这样可以清楚表达它所代表的元素。

例 3-10　类名模糊的声明块

```
.a {
  width: 200px;
}
```

然而，虽然将类名表意清楚很重要，但同样重要的是不要做过头。如果不用 `animal` 作为类名，而是用 `female-black-and-white-kitten`（雌性—黑—和—白—小猫）又将如何？从技术上来讲，新类名可以重用，但是它过于精确，因为其他动物也许会用同一样式来显示，那么该类名就不能准确表述它所应用于的动物。相比之下，类名 `.animal` 则更加贴切，因为它能表达清楚意思，看到它很容易理解为什么元素增加样式。它还具有一定的概括性，能够表达应该应用该样式的任何类型动物，不论雌雄、老幼，或者是否是猫。

避免使用过于模块化的类

有意义的类名，描述的是应用样式的元素，而不是为元素应用的样式。你是否见过类似例 3-11 这样的 HTML ？

例 3-11　过于模块化的类

```
<h1 class="font-bold uppercase blue-text margin-bottom-large no-padding">
  Too Many CSS Classes
</h1>
```

这些类描述的是**怎样**为元素增加样式，而不是为**什么元素**增加样式。此外，这些类存在**过于模块化问题**——因为每个类应用一种样式，它们总是需要一起使用。应该避免使用过于模块化的类，因为它们并不比行内样式更好，如例 3-12 所示。

例 3-12　行内 CSS

```
<h2 style="font-weight: bold; text-transform: uppercase; color: #1200FF;
 margin-bottom: 20px; padding: 0">
  Too Many CSS Classes
</h2>
```

这些样式应该放到一起，使用表意明确、能够描述为什么元素增加样式的类名，请见例 3-13。按照该方式重构之后，HTML 可读性更强，因为它使用了非常简洁的类名，如例 3-14 所示。

例 3-13　描述为什么元素增加样式的类

```
.section-title {
  color: #1200FF;
  font-weight: bold;
  margin-bottom: 20px;
  padding: 0;
  text-transform: uppercase;
}
```

例 3-14　将过于模块化的类组织在一起的类

```
<h2 class="section-title">
  Too Many CSS Classes
</h2>
```

3.7　创建更好的盒子

盒子模型是浏览器决定如何渲染矩形的方法。理解盒子模型的工作原理很重要，因为所有的 HTML 元素本质上都是盒子，只有掌握其原理，才能将各元素合理地组织在一起。

图 3-1 展示了一个具有固定宽、高、内边距（padding）、外边距（margin）和边框（外边距和内边距使用了极具区分度的颜色，很直观）的元素。该盒子可用例 3-15 中的代码生成。

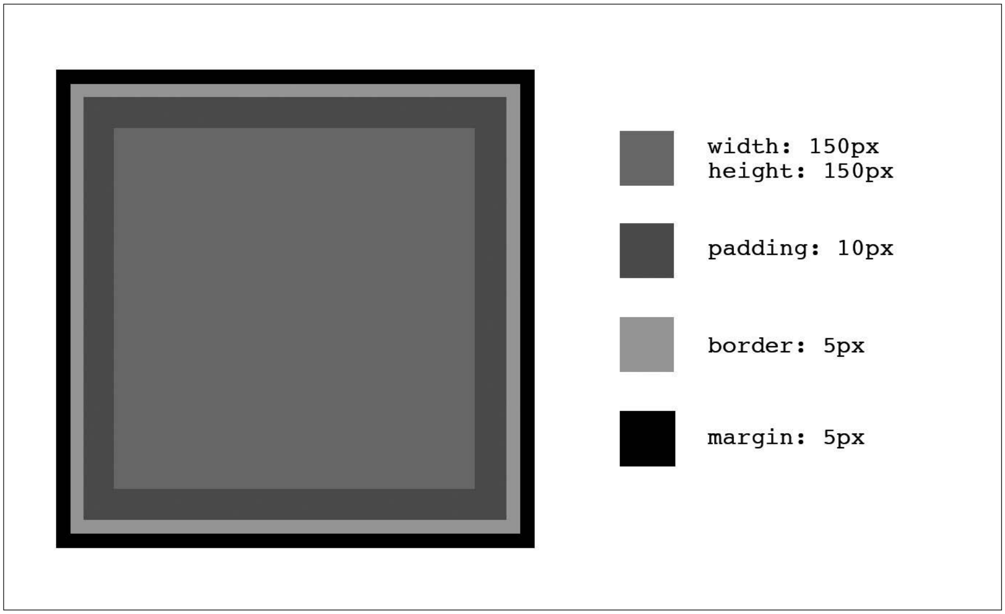


图 3-1：具有固定高、宽、内边距、外边距和边框的元素

例 3-15　图 3-1 所示盒子的代码

```
<!doctype html>
<html>
  <head>
    <title>Determining Dimensions with the Box Model</title>
    <style type="text/css">
      .example-element {
        background-color: #FF0000;
        border: 5px solid #000000;
        display: block;
        height: 150px;
        margin: 5px;
        padding: 10px;
        width: 150px;
      }
    </style>
  </head>
  <body>
    <div class="example-element"></div>
  </body>
</html>
```

根据为元素的 box-sizing 属性赋的值，盒子尺寸的计算方式有两种。在下面两种情况中，外边距都将影响盒子周边的空间，但是计算盒子的尺寸时，不需要考虑外边距。

3.7.1 盒子尺寸：content-box

若将 `content-box` 赋给 `box-sizing` 属性，在计算盒子的尺寸时，需要为元素的高度和宽度增加内边距和边框。例如，如果按照这种方式计算图 3-1 盒子的尺寸，则为 180px × 180px，因为：

```
  150px 高度
+  10px padding-top
+  10px padding-bottom
+   5px border-top
+   5px border-bottom = 180px 计算后得到的高度

  150px 宽度
+  10px padding-left
+  10px padding-right
+   5px border-left
+   5px border-right = 180px 计算后得到的宽度
```

3.7.2 盒子尺寸：border-box

若将 `border-box` 赋给 `box-sizing` 属性，盒子的尺寸只跟盒子的宽度和高度属性相关。这表明尽管图 3-1 为盒子定义了内边距和边框样式，盒子的大小仍为 150px 高、150px 宽，因为这就是为高度和宽度属性设置的值。浏览器会考虑内边距和边框，合理调整高度和宽度属性，因此总尺寸等于为高度和宽度属性设置的值。在这种情况下：

```
  150px 计算后得到的高度
-  10px padding-top
-  10px padding-bottom
-   5px border-top
-   5px border-bottom = 120px 剔除内边距和边框，盒子的隐性高度

  150px 计算后得到的宽度
-  10px padding-left
-  10px padding-right
-   5px border-left
-   5px border-right = 120px 剔除内边距和边框，盒子的隐性宽度
```

3.7.3 content-box或border-box

因为盒子尺寸有两种不同的计算方法，所以应该考虑其使用场景。`content-box` 和 `border-box` 两者没有优劣之分，但是很多人发现 `border-box` 更直观，因为它描述的是包括边框在内的元素的高度和宽度，而不只是内容区域的尺寸。

任何元素都可以设置 `box-sizing` 属性，因此可以混用这两种盒子，但是为了保持一致性，通常选用其中一种并坚持使用。具体设置方法是，用通用选择器进行设置，指定盒子的类型：

```
*,
*:after,
*:before {
  box-sizing: border-box;
}
```

3.8　总结

本章所讲概念为我们进行重构打下了另一个基础。重构之前，理解如何编写更优质的 CSS，将会使得重构更加容易。下一章，我们将探索样式的不同用途及其对代码复用有何帮助。学习下一章时，不要忘记本章内容，因为灵活运用所学知识可极大地简化 CSS 代码。

第 4 章

为样式分类

代码复用是优秀架构的一项基本原则，可以说它是编写高质量 CSS 代码所要重点考虑的原则之一。本章讨论的是，以符合逻辑、深思熟虑的方式作用于 HTML 元素的不同样式，在用途上有哪些细微差别。按照样式的功能对其进行分类和使用，代码的复用方式就会变得更加明显。学习本章过程中，请回顾第 2 章所讲内容，你会发现为样式分类跟级联样式之间的关系。

4.1　样式分类的重要性

简单来说，网站是一组用来展示信息的文档。然而，往复杂处说，最复杂的网站很像一个功能多种多样的应用，帮助实现简单的人机交互和复杂的操作。这两种极端，都使用语义化 HTML 标签描述网站所展示的内容，不同复杂程度的网站，都可以从按用途定义样式的方法中受益。

按用途定义样式，有助于创建更优秀的架构，因为将样式组织成为不同的类别，促使代码可预测性更强、更易于复用。接下来所介绍的多种样式分类方法尽管很复杂，却适用于任何网站。

4.2　通用样式

浏览器自带的默认样式表叫作**浏览器默认样式**，它可为 HTML 元素应用默认的样式。因为不同厂商开发的浏览器不同，所以其默认样式表的某些属性和属性值可能有所差异。

通用样式是指为各种元素的属性设置默认值的样式，否则不同的浏览器将为其应用不同的样式。例如，例 4-1 规范了 `<hr>` 元素在不同浏览器中的样式。

例 4-1　规范 `<hr>` 元素在不同浏览器中的样式

```
/**
 * 1. 增加在Firefox浏览器中应用的盒子尺寸类型。
 * 2. 在Edge和IE浏览器中，显示溢出部分。
 */

hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}
```

测试和记录浏览器默认设置的属性和属性值很难，但幸运的是 Web 开发社区已经为我们做了大量工作，因此有多种开源的通用样式可供选择。最通用的样式表是 Nicolas Gallagher 和 Jonathan Neal 开发的 normalize.css（例 4-1 中的 CSS 即摘自该文件）。你可以从 GitHub（http://necolas.github.io/normalize.css）下载，附录也附了一份。

开源通用样式表中的很多样式，它们的最大用途体现在解决传统浏览器的兼容问题方面，因此你的项目很可能没必要使用它们（取决于项目所支持的浏览器）。你可能也会发现其中包含大量你用不到的元素的样式，比如 `<audio>`、`<canvas>` 和 `<kbd>` 等。如果你不打算使用这些元素，为了使 CSS 文件缩小，应该考虑将其删除。

4.3　基础样式

基础样式旨在为设置更加细致的样式提供基础。基础样式很容易识别，因为它们用单类型选择器，或结合使用非常简单的类型选择器和结合符（例如，用 `ul ul` 在无序列表中选择无序列表），或使用任何伪类为 HTML 元素添加样式。跟通用样式一样，基础样式是最不精确的样式，应被置于样式表中。

一旦为 HTML 元素设置基础样式后，应该不需要重新声明，除非设置的基础样式不适合另一目标场景。编写基础样式所要遵循的基本原则是：为元素应用基础样式之外的其他样式时，不需要重写大量基础样式就能实现设计目标。

4.3.1　定义基础样式

下面是如何为不同类型常用元素定义基础样式的一些建议。其中多个属性，通常在浏览器样式表中有所设置，但其属性值也许不适合所有设计目标，并且随着浏览器发布新版本，这些值可能会发生变化。为元素定义基础样式之前，可以思考一下样式的主要应用场景，并以此作为定义样式的指导。

基础样式应该只为最笼统的使用场景设置属性和属性值。通常我们为元素设置以下基础属性：

- color
- font-family
- font-size
- font-weight
- letter-spacing
- line-height
- margin
- padding

如果你的网站本质上是信息类网站，用好这些样式可能就足够了。然而，如果你搭建的是设计非常复杂的应用类网站，那么这些样式只能满足初级需求。正如本章后面所讲，对于可复用的组件，也许需要更复杂的样式。

你在编写基础样式时，需考虑上面列出的这些属性，但并不是任何时候都要全部设置，因为它们都继承自祖先元素（除了 margin 和 padding）。如果 margin 和 padding 应该使用继承来的属性值，则用 inherit 作为属性值。对于某种特定类型的元素，应该纳入基础样式的其他属性或伪类，这个在下面几节将会介绍。

利用继承

对于 color、font-family、font-size、font-weight、letter-spacing 和 line-height 属性，子元素继承自父元素，因此子元素的属性值不总是需要设置。关于这些 CSS 属性值是否继承的完整列表，请见 https://www.w3.org/TR/CSS21/propidx.html。需要指定样式的完整 HTML 元素列表，请见 https://www.w3.org/TR/html-markup/elements.html。

4.3.2　文档元数据元素

记录元数据的标签包括 <head>、<title>、<base>、<link> 和 <meta>。因为它们不可见，故不能为其添加样式。

4.3.3　区块元素

区块元素包括 <address>、<article>、<aside>、<body>、<footer>、<header>、<nav> 和 <section>。该类元素通常包含其他元素，它们组成了 HTML 文档的各种区域。

考虑一下为区块元素设置以下属性：

- color
- font-family
- font-size
- font-weight
- letter-spacing
- line-height
- padding

<body> 元素也许还需要设置背景色 background 属性，例 4-2 展示了如何为区块元素应用基础样式。

例 4-2　为区块元素设置基础样式

```
body {
  background: #FFFFFF;
  color: #333333;
  font-family: Helvetica, Arial, sans-serif;
  font-size: 14px;
  line-height: 1.3;
  padding: 5% 20%;
}

article,
footer,
header,
nav {
  padding: 0;
}

article,
nav {
  margin-bottom: 12px;
  margin-top: 12px;
}

footer {
  margin-top: 12px;
}

header {
  margin-bottom: 12px;
}
```

4.3.4　标题和文本元素

标题元素包括 <h1>—<h6> 六级标题，用于定义 HTML 文档每个区域的标题。文本元素包括 <figure>、<figcaption>、<p> 和 <pre>，用来展示块状文本。

为标题和文本元素定义基础样式时，需考虑以下属性：

- font-family
- font-size
- font-weight
- letter-spacing
- line-height
- margin-bottom
- margin-top

例 4-3 展示了如何为标题和文本元素定义基础样式。

例 4-3　标题和文本元素的基础样式

```
h1,
h2,
h3,
h4,
h5,
h6 {
  font-family: Georgia, Times, serif;
  font-weight: 100;
  line-height: 1.1;
  margin: 0.5em 0;
}

h1 {
  font-size: 36px;
}

h2 {
  font-size: 24px;
}

h3 {
  font-size: 21px;
}

h4 {
  font-size: 18px;
}

h5 {
  font-size: 16px;
}

h6 {
  font-size: 14px;
}

p,
pre {
  margin-bottom: 12px;
  margin-top: 12px;
}
```

4.3.5 锚点标签元素

锚点标签为其他 HTML 文档或同一 HTML 文档的其他区域提供链接。它们常用 :link、:visited、:focus、:hover 和 :active 伪类展示状态，因此为其定义基础样式时，不要忘记这些伪类。下面一一说明这些伪类的用途。

- :link 样式应用于具有合法 href 属性的元素。
- :visited 样式应用于具有合法 href 属性的超链接元素，并且浏览器的历史记录之中含有该链接。
- :focus 样式应用于获得焦点的超链接元素。当点击、轻触元素，或使用 Tab 键跳转到当前元素时，元素获得焦点。
- :hover 样式应用于鼠标光标悬浮于超链接之上时。触摸设备,没有光标悬浮状态。因此，通常当轻触元素时，应用 :hover 样式；轻触其他元素时，移除 :hover 样式。
- :active 样式应用于超链接被“激活”时。若使用鼠标，这一时刻发生在点击超链接，鼠标按键弹起前。在触摸设备上，这一时刻发生在轻触元素，手指移开之前。

还需要注意的是，如果使用伪类，:link 和 :visited 这两个伪类是首先要指定的。所有这些伪类的特指度相等，因此级联方法将根据定义的顺序应用它们。这表明，如果一条超链接被访问，并且 :visited 是在 :hover、:focus 或 :active 后面定义的，将优先使用前面为 :visited 伪类定义的、与后面重复的样式。

另一个需要记住的事项是，所有这些伪类有效地给予超链接更高的特指度，因为它们由类型选择器和伪类选择器组成。这表明要覆盖这些样式，需要更高特指度或相同特指度，但在样式表中处于偏后位置的样式。因此，:link 伪类常常被忽略，而超链接应该使用的样式直接用 a 类型选择器来设置。

为锚点标签及其伪类定义基础样式时，应该考虑的常用属性包括：

- background-color
- border
- color
- font-weight
- text-decoration

虽然浏览器自带的默认样式就足够了，但是为 :focus 伪类添加样式时，为其定义 outline-width、outline-style 和 outline-color 属性（或更简洁的 outline 属性）更加方便。

> 若从超链接元素的 :foucs 伪类中删除 outline 属性，且不在页面上以可见的形式加以提示，对于只用键盘或使用其他交互方式有别于鼠标的用户而言，严重影响网站的可用性。

锚点标签用于修改文本或其他行内元素，因此通常将其默认为行内元素[1]。我们还可以为锚点标签设置 font-family、font-size 和 font-weight 属性，但是很可能其属性值将继承祖先元素。例 4-4 展示的是如何为锚点标签元素定义基础样式。

例 4-4　超链接的基础样式

```
a,
a:visited,
a:focus,
a:hover,
a:active {
  color: inherit;
  text-decoration: underline;
}

a:hover {
  background-color: #FFFF00;
}
```

4.3.6　文本语义元素

文本语义元素是指为文本提供更多含义或结构的元素。这些元素通常为行内元素，其中包括 <abbr>、<b>、<cite>、<code>、<data>、<dfn>、<em>、<i>、<kbd>、<s>、<strong>、<sub>、<sup>、<time> 和 <u> 等标签。

因为该类元素是用来修改文本的样式，因此在为其定义基础样式时，应考虑如下属性：

- color
- font-family
- font-size
- font-weight

例如，我们可以为 <code> 标签定义如下样式：

```
code {
  color: #00FF00;
  font-family: monospace;
  font-weight: 500;
  line-height: 1.5;
}
```

4.3.7　列表

列表元素包括 （有序列表）、<ul>（无序列表）和 <dl>（定义列表）元素。有序和无序列表内部仅能直接包含 （列表项目）元素，定义列表元素仅能直接包含 <dt>（定义

注 1：也可称为内联元素。——译者注

术语）和 <dd>（定义描述）元素。

因为列表有很多种，所以为列表定义合适的基础样式有一定难度。如果网站以展示信息为主，通常使用有序列表、项目符号列表或带有缩进的定义列表；如果网站用户界面更加复杂，列表的应用场景有多种，则需要根据相应的设计添加样式。列表的一些应用场景包括水平导航、商品列表、社交媒体用户的个人主页，等等。

为有序和无序列表元素定义基础样式时，应该考虑以下属性：

- font-family
- font-size
- list-style-type 或 list-style-image
- list-style-position
- line-height
- margin-bottom
- margin-top
- padding-left

对于纯粹的信息类网站，也许需要设置 list-style-type 或 list-style-image 和 list-style-position 属性，辅助展示列表的各个项目。然而，对于更像是应用类的网站，如果较少使用它们，最好为其赋 none 值，从而不必每次都重写其样式。为了防止子元素的缩进， 或 <ul> 元素的 padding-left 属性值应该设置为 0。子元素 <li> 从父元素 <ol> 或 <ul> 继承 font-family、font-size 和 line-height 属性，但是不继承 margin 或 padding 属性。

我们可以为无序列表指定以下样式：

```
ul {
  list-style-position: outside;
  list-style-type: disc;
  margin-top: 0;
  margin-bottom: 12px;
}


ul ul {
    margin-bottom: 0;
}
```

虽然因 list-style-type、list-style-image 或 list-style-position 只是应用于 display 属性设置为 list-item 的元素，而导致它们不是很常用，但是有序和无序列表具有的属性，定义列表同样具有。list-item 是 li 元素 display 属性的默认值，而 <dt> 和 <dd> 元素该属性的默认值是 block。

<dd> 和 <dt> 元素从其父元素 <dl> 继承 font-family、font-size 和 line-height 属性。我们通常为 <dd> 元素设置 margin-left 属性来实现缩进效果（若避免由此带来的缩进，可将该属性值设置为 0）。

4.3.8 组合元素

组合元素（grouping element）包括 <div>、<main> 和 <span>。虽然，从技术上讲，<span> 标签是文本级的语义标签，但是其主要用途是将文本或行内元素包在一起。

<div> 和 <main> 标签是典型的块状元素，而 <span> 标签是行内元素。该类元素用于组合其他标签，因此通常没必要为其定义基础样式，它们的样式根据具体情况用类来定义。然而，如果 <main> 标签用作可见的容器，最好为其设置 margin 和 padding 属性。

4.3.9 表格

表格，顾名思义，是用表格状结构展示数据。以表格状结构展示数据，需要用到的元素包括 <table>、<caption>、<colgroup>、<col>（列）、<tbody>（表格主体）、<thead>（表头）、<tfoot>（表格的页脚）、<tr>（表格行）、<td>（表格单元格）和 <th>（表头单元格）元素。20 世纪 90 年代至 21 世纪初，创建页面布局普遍使用表格，但是由于现在 CSS 和浏览器已经成熟，因此更合理的做法是用其他元素创建布局，而只用表格展示表格状数据。

为 <table> 元素定义基础样式时，需要考虑以下属性：

- border-collapse
- border-spacing
- border（border-width、border-color 和 border-style）
- empty-cells
- font-family
- font-size
- letter-spacing
- line-height

为 <thead>、<tbody> 和 <tfoot> 元素定义基础样式时，需要考虑以下属性：

- background-color
- color
- text-align
- vertical-align

为 <th> 和 <td> 元素定义基础样式时，需要考虑以下属性：

- background-color
- border（border-width、border-color 和 border-style）
- color
- font-family
- font-size
- letter-spacing
- line-height
- text-align
- vertical-align

综合以上内容，也许应该为表格定义类似下面的样式：

```
table {
  border-collapse: collapse;
  border-spacing: 0;
  empty-cells: show;
  border: 1px solid #000000;
}

tfoot,
thead {
  text-align: left;
}

thead {
  background-color: #ACACAC;
  color: #000000;
}

th,
td {
  border-left: 1px solid #000000;
  padding: 0.5em 1em;
}

th:first-child,
td:first-child {
  border-left: none;
}
```

4.3.10 表单

表单用来收集用户信息，表单元素包括 <form>、<label>、<input>、<button>、<select>、<datalist>、<optgroup>、<option>、<textarea>、<output>、<progress>、<meter>、<fieldset> 和 <legend>。为该类元素定义基础样式时，应考虑以下属性：

- font-family
- font-size
- line-height
- margin
- padding

对于普通的设计，为 <form> 元素设置样式，子元素从它继承即可，但是更为复杂的设计，也许需要更多样式。<form> 的子元素 <legend>、<label> 和 <input> 的 font-weight、font-size 和 font-family 属性通常不同于父元素 <form>，因此应该在子元素上定义这三个属性。

有一些表单元素，我们很难为其添加样式，因为很多浏览器会忽略应用于它们的属性。例如，浏览器会忽略为复选框和单选按钮的 border-color、border-width、background-color 和许多其他属性定义的样式。解决办法是，自定义复选框和单选按钮组件，隐藏表单的控制控件，并使用其他 HTML 元素实现按钮效果，但这不是基础样式所要实现的。

为表单元素定义以下样式，也许比较合理：

```
fieldset {
  border: 0;
  margin: 0;
  padding: 0;
}

input {
  display: block;
  font-size: inherit;
  padding: 4px;
  width: 100%;
}

label {
  display: block;
  font-weight: 900;
  margin-bottom: 6px;
  padding: 4px;
}

legend {
  border: 0;
  color: #000000;
  display: block;
  font-size: 1.2em;
  margin-bottom: 12px;
  padding: 0 12px;
  width: 100%;
}
```

4.3.11 图像

图像可以用 `<img>` 或 `<picture>` 标签展示。为图像元素定义基础样式时，应该考虑如下属性：

- border
- max-width
- vertical-align

因为 <img> 元素可以用在行内元素之中，vertical-align 属性的默认值为 baseline，这也许适合或不适合你的设计任务。此外，当 <img> 元素置于设置了大小的块状元素之中时，将 max-width 属性设置为父容器的 100%，能够防止图像溢出容器。

img 标签的基础样式可以定义为：

```
img {
  border: none;
  max-width: 100%;
  vertical-align: middle;
}
```

4.4 组件样式

可复用组件是指添加了样式的元素或元素组合利用视觉隐喻，使得用户与网站的交互更加容易。可复用组件的例子包括按钮、下拉菜单、模态窗口、进度条和选项卡。

可复用组件易于辨识，但是正确创建它们有一定难度。创建可复用组件之前，思考以下问题将对你很有帮助。

- 只有一个组件，还是有一个以上的组件组合在一起？
- 组件是行内元素、块状元素还是其他类型（例如：组件以绝对定位的方式独立于文档流吗）？

对以上问题的答案做到心中有数后，创建可复用组件的过程可以简化如下。

(1) 创建组件之前，定义需要实现的行为。
(2) 保持组件样式的粒度，设置合理的默认值。
(3) 若需要重写组件组的可见样式，用容器元素将它们包起来，为该容器定义一个具有区别度的类。
(4) 将定义元素尺寸的任务交给结构化容器。

下面示例解释了如何构建一个简单的选项卡组件（图 4-1），请结合该示例逐条探索上述指南。

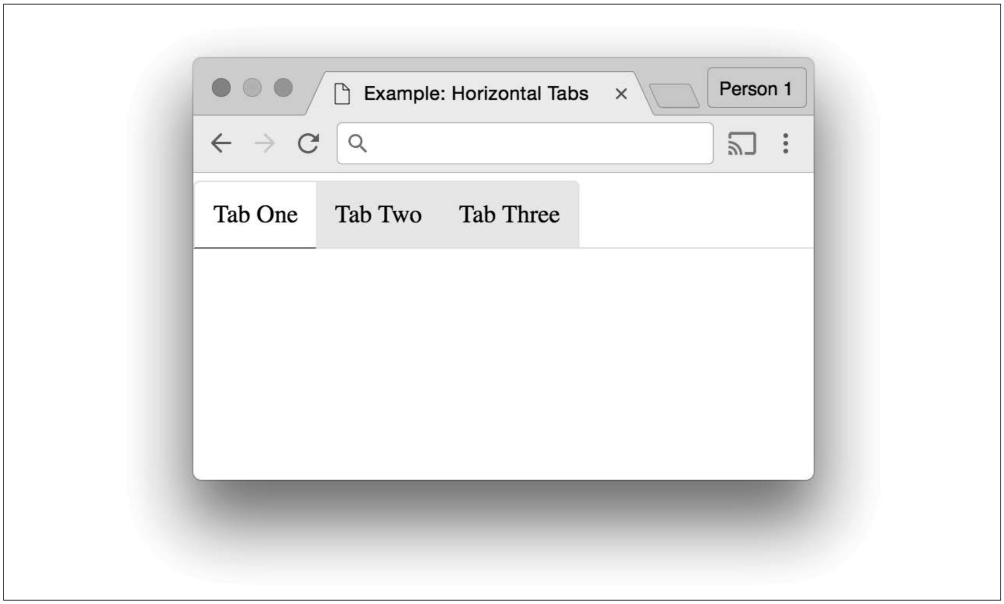


图 4-1：选项卡组件

4.4.1　定义需要实现的行为

对于这个示例，我们要创建如图 4-1 所示的选项卡组件。需求如下。

- 图 4-1 的选项卡组件有三个选项卡，但是该种组件应该能够支持两个及以上选项卡。
- 选项卡激活时，该选项卡的底边框变为蓝色，背景变为白色。
- 选项卡处于非激活状态时，背景为灰色。

我们使用例 4-5 所示的 HTML 代码创建选项卡组件。

例 4-5　有三个选项卡的选项卡组件代码

```
<nav>
  <ul>
    <li><a href="#">Tab One</a></li>
    <li><a href="#">Tab Two</a></li>
    <li><a href="#">Tab Three</a></li>
  </ul>
</nav>
```

4.4.2　保持组件样式的粒度

保持组件样式的粒度是指为组件的每个元素添加样式，以便它们能够复用。例 4-5 中有三个选项卡，因此应该为每个选项卡（例 4-6）创建一个类，但是这将引入大量重复代码。

例 4-6　为每个选项卡定义一个单独的类

```
/**
 * 为每个选项卡定义一个单独的类，产生大量重复代码。
 */
.tab-1 { /* styles go here */ }
.tab-2 { /* styles go here */ }
.tab-3 { /* styles go here */ }

.tab-1:hover { /* styles go here */ }
.tab-2:hover { /* styles go here */ }
.tab-3:hover { /* styles go here */ }

.tab-1.active { /* styles go here */ }
.tab-2.active { /* styles go here */ }
.tab-3.active { /* styles go here */ }

.tab-1 > a { /* styles go here */ }
.tab-2 > a { /* styles go here */ }
.tab-3 > a { /* styles go here */ }

.tab-group { /* styles go here */ }
```

上述实现方式不太优雅——每个类的声明块跟其他类相同，除去客户端需要下载更大的 CSS 文件，还会增加显示效果不一致的风险，因为需要维护的代码相应增多。为了避免这些问题，我们可以按照例 4-7 的做法将每个选项卡的类组合起来，但是仍然有很多类实现的效果相同。

例 4-7　将每个选项卡的类组合起来

```
/**
 * 像下面这样将每个选项卡的类组合起来，产生大量没有用处的重复代码。
 */

.tab-1,
.tab-2,
.tab-3 {
  /* 将样式写到这里 */
}

.tab-1:hover,
.tab-2:hover,
.tab-3:hover {
  /* 将样式写到这里 */
}

.tab-1.active,
.tab-2.active,
.tab-3.active {
  /* 将样式写到这里 */
}

.tab-1 > a,
.tab-2 > a,
```

```
.tab-3 > a {
  /* 将样式写到这里 */
}

.tab-group {
  /* 将样式写到这里 */
}
```

最佳的解决方案是将选项卡的样式抽象为一个可复用的类（例 4-8）。请注意，例 4-8 还定义了 `.active` 类，我们可以根据选项卡状态添加它，以表示哪个选项卡处于激活状态。此外，下述代码还定义了 `.tab-group` 元素的样式，该元素包含多个选项卡。

例 4-8　选项卡样式

```
/**
 * 选项卡组件样式
 */

.tab {
  background-color: #F2F2F2;
  border-bottom: 1px solid #EEEEEE;
  border-top: 1px solid #EEEEEE;
  bottom: -1px;
  display: inline-block;
  margin-left: 0;
  margin-right: 0;
  margin-top: 4px;
  position: relative;
}

.tab:first-child {
  border-left: 1px solid #EEEEEE;
  border-top-left-radius: 4px;
}

.tab:last-child {
  border-right: 1px solid #EEEEEE;
  border-top-right-radius: 4px;
}

.tab.active {
  background-color: #FFFFFF;
  border-bottom: 1px solid #2196F3;
  color: #000000;
}

.tab:hover {
  background-color: #F9F9F9;
}

.tab > a {
  color: inherit;
  display: block;
  height: 100%;
```

```
  padding: 12px;
  text-decoration: none;
  width: 100%;
}

/**
 * 选项卡组件容器
 */

.tab-group {
  border-bottom: 1px solid #EEEEEE;
  list-style: none;
  margin: 0;
  padding-left: 0;
}
```

既然已经为选项卡组件的各个类定义了恰当的样式，接下来我们可以用这些类来更新选项卡组件的 HTML 代码，请见例 4-9。

例 4-9　用新选项卡类更新包含三个选项卡的 HTML 代码

```
<nav>
  <ul class="tab-group">
    <li class="tab active"><a href="#">Tab One</a></li>
    <li class="tab"><a href="#">Tab Two</a></li>
    <li class="tab"><a href="#">Tab Three</a></li>
  </ul>
</nav>
```

4.4.3　根据需要，改写元素容器的样式

至此，我们实现了一个水平选项卡组件，但是如果我们想垂直展示选项卡该怎么做呢？如要为选项卡应用不同样式，我们可将定义选项卡样式的任务交由父容器完成。为了实现这一功能，首先创建一个新的 `.tab-group-vertical` 类，该类包含一个 `.tab` 元素，然后用 `.tab-group` 类将 `.tab` 元素组合在一起：

```
.tab-group,
.tab-group-vertical {
  list-style: none;
  margin: 0;
  padding-left: 0;
}
```

接下来，我们将之前在 `.tab` 规则集中定义的一些声明挪到用更加精确的选择器定义的规则集中，以便新定义的几个类不用重复改写这些规则集。完成上述改动后，`.tab` 类将只包含适用于所有选项卡的样式：

```
.tab {
  background-color: #F2F2F2;
  margin-left: 0;
  margin-right: 0;
```

```
  position: relative;
}

.tab:hover {
  background-color: #F9F9F9;
}

.tab.active {
  background-color: #FFFFFF;
  color: #000000;
}
```

现在，应该将 border、border-radius 和 display 样式交由合适的父容器，.tab-group、.tab-group-vertical 类各有各的边框样式（例 4-10）。

例 4-10　将属性交由合适的容器

```
/**
 * 水平选项卡组
 */

.tab-group {
  border-bottom: 1px solid #EEEEEE;
}

.tab-group .tab {
  border-bottom: 1px solid #EEEEEE;
  border-top: 1px solid #EEEEEE;
  bottom: -1px;
  display: inline-block;
}

.tab-group .tab:first-child {
  border-left: 1px solid #EEEEEE;
  border-top-left-radius: 4px;
}

.tab-group .tab:last-child {
  border-right: 1px solid #EEEEEE;
  border-top-right-radius: 4px;
}

.tab-group .tab.active {
  border-bottom: 1px solid #2196F3;
}

/**
 * 垂直选项卡组
 */

.tab-group-vertical {
  border-left: 1px solid #EEEEEE;
}
```

```
.tab-group-vertical .tab {
  border-left: 1px solid #EEEEEE;
  border-right: 1px solid #EEEEEE;
  left: -1px;
  display: block;
}

.tab-group-vertical .tab:first-child {
  border-top: 1px solid #EEEEEE;
  border-top-right-radius: 4px;
}

.tab-group-vertical .tab:last-child {
  border-bottom: 1px solid #EEEEEE;
  border-bottom-right-radius: 4px;
}

.tab-group-vertical .tab.active {
  border-left: 1px solid #2196F3;
}
```

上述代码还有一处需要重构，你也许已经注意到了，`1px solid #EEEEE` 语句重复出现了多次。我们可以通过在 `.tab` 类中定义 `border-color` 和 `border-style` 属性，合理设置 `border-width` 属性值，必要时重写 `border-color` 属性对此进行重构，如例 4-11 所示。该方法的优点是，若要修改选项卡颜色，所需改动之处更少。

例 4-11　用新选项卡类更新包含三个选项卡的 HTML 代码

```
/**
 * 选项卡组件样式
 */

.tab {
  background-color: #F2F2F2;
  margin-left: 0;
  margin-right: 0;
  position: relative;
}

.tab:hover {
  background-color: #F9F9F9;
}

.tab.active {
  background-color: #FFFFFF;
  color: #000000;
}

.tab > a {
  color: inherit;
  display: block;
  height: 100%;
```

```
  padding: 12px;
  text-decoration: none;
  width: 100%;
}

/**
 * 选项卡组件容器
 */

.tab-group,
.tab-group-vertical {
  list-style: none;
  margin: 0;
  padding-left: 0;
}

.tab,
.tab-group,
.tab-group-vertical {
  border-color: #EEEEEE;
  border-style: solid;
  border-width: 0;
}

/**
 * 水平选项卡组
 */

.tab-group {
  border-bottom-width: 1px;
}

.tab-group .tab {
  border-bottom-width: 1px;
  border-top-width: 1px;
  bottom: -1px;
  display: inline-block;
}

.tab-group .tab:first-child {
  border-left-width: 1px;
  border-top-left-radius: 4px;
}

.tab-group .tab:last-child {
  border-right-width: 1px;
  border-top-right-radius: 4px;
}

.tab-group .tab.active {
  border-bottom-color: #2196F3;
  border-bottom-width: 1px;
}
```

```
/**
 * 垂直选项卡组
 */

.tab-group-vertical {
  border-left-width: 1px;
}

.tab-group-vertical .tab {
  border-left-width: 1px;
  border-right-width: 1px;
  left: -1px;
  display: block;
}

.tab-group-vertical .tab:first-child {
  border-top-width: 1px;
  border-top-right-radius: 4px;
}

.tab-group-vertical .tab:last-child {
  border-bottom-width: 1px;
  border-bottom-right-radius: 4px;
}

.tab-group-vertical .tab.active {
  border-left-color: #2196F3;
  border-left-width: 1px;
}
```

用例 4-11 所示的 CSS 代码渲染一组用 `.tab-group-vertical` 作为类的选项卡，将得到图 4-2 所示的效果。

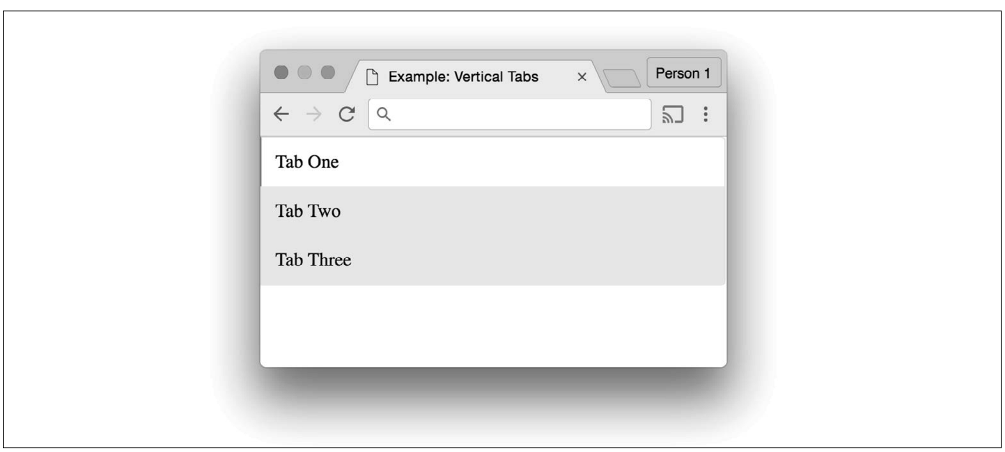


图 4-2：垂直选项卡

这些选项卡显示效果还存在问题，我们接下来将定义选项卡尺寸的任务交给结构化容器。

4.4.4　将定义尺寸的任务交给结构化容器

你也许已经注意到了，我们前面编写的 `.tab-group` 和 `.tab-group-vertical` 组件没有指定尺寸。其实是有意省略的，因为我们接下来要将定义尺寸的任务交给包着组件或组件组的外层结构。下一节要讲的**结构化样式**，指的是支配页面基本结构的样式。你可以创建任意不同类型的布局，比如带侧边栏、分栏或其他任何你能想到的布局。

组件能够复用固然很好，但是我们很难预测它出现在页面中的位置。这也正是为什么要将设置元素尺寸的任务交给盛放组件的元素。

例如，假定我们希望展示两组水平选项卡，每组各占桌面浏览器 50% 的可视区域，如图 4-3 所示。我们可以在 `.tab-group` 定义中增加 `width: 50%` 的声明，但如果还需要在其他不同位置使用选项卡，像前面这样定义宽度，复用性不高。因而，我们需要像例 4-12 那样，将设置尺寸的任务交给类名为 `.tabbed-pane` 的结构化元素。该例纯粹是为了说明如何将定义尺寸的任务交给其他元素，实际应用中你可能不想在同一页面使用多组不同的选项卡。

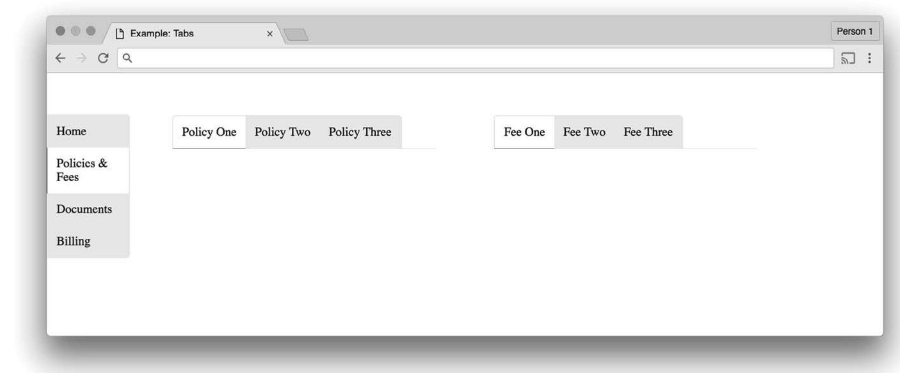


图 4-3：将定义组件尺寸的任务交给结构化容器

例 4-12　将定义组件尺寸的任务交给结构化容器的代码

```
<!doctype html>
<html>
  <head>
    <title>Example: Tabs</title>
    <style type="text/css">

*,
*:after,
*:before {
  box-sizing: border-box;
```

```
}

body {
  margin: 0;
  padding: 0;
}

/**
 * 选项卡组件样式
 */

.tab {
  background-color: #F2F2F2;
  margin-left: 0;
  margin-right: 0;
  position: relative;
}

.tab:hover {
  background-color: #F9F9F9;
}

.tab.active {
  background-color: #FFFFFF;
  color: #000000;
}

.tab > a {
  color: inherit;
  display: block;
  height: 100%;
  padding: 12px;
  text-decoration: none;
  width: 100%;
}

/**
 * 选项卡组件容器
 */

.tab-group,
.tab-group-vertical {
  list-style: none;
  margin: 0;
  padding-left: 0;
}

.tab,
.tab-group,
.tab-group-vertical {
  border-color: #EEEEEE;
  border-style: solid;
  border-width: 0;
}
```

```
/**
 * 水平选项卡组
 */

.tab-group {
  border-bottom-width: 1px;
}

.tab-group .tab {
  border-bottom-width: 1px;
  border-top-width: 1px;
  bottom: -1px;
  display: inline-block;
}

.tab-group .tab:first-child {
  border-left-width: 1px;
  border-top-left-radius: 4px;
}

.tab-group .tab:last-child {
  border-right-width: 1px;
  border-top-right-radius: 4px;
}

.tab-group .tab.active {
  border-bottom-color: #2196F3;
  border-bottom-width: 1px;
}

/**
 * 垂直选项卡组
 */

.tab-group-vertical {
  border-left-width: 1px;
}

.tab-group-vertical .tab {
  border-left-width: 1px;
  border-right-width: 1px;
  left: -1px;
  display: block;
}

.tab-group-vertical .tab:first-child {
  border-top-width: 1px;
  border-top-right-radius: 4px;
}

.tab-group-vertical .tab:last-child {
  border-bottom-width: 1px;
  border-bottom-right-radius: 4px;
}
```

```
.tab-group-vertical .tab.active {
  border-left-color: #2196F3;
  border-left-width: 1px;
}

/**
 * 选项卡组件容器
 */

.tabbed-pane {
  display: block;
  width: 100%;
}

.tabbed-pane .tab-group {
  float: left;
  width: 45%;
}

.tabbed-pane .tab-group:first-child {
  margin-right: 5%;
}

.tabbed-pane .tab-group:last-child {
  margin-left: 5%;
}

/**
 * 结构化样式
 */

.global-nav {
  float: left;
  padding: 5% 0;
  width: 10%
}

.content {
  float: left;
  padding: 5%;
  width: 80%;
}

  </style>
</head>
<body>
  <nav class="global-nav">
    <ul class="tab-group-vertical">
      <li class="tab"><a href="#">Home</a>
      <li class="tab active"><a href="#">Policies &amp; Fees</a>
      <li class="tab"><a href="#">Documents</a>
      <li class="tab"><a href="#">Billing</a>
    </ul>
  </nav>
```

```
    <main class="content">
      <nav class="tabbed-pane">
        <ul class="tab-group">
          <li class="tab active"><a href="#">Policy One</a>
          <li class="tab"><a href="#">Policy Two</a>
          <li class="tab"><a href="#">Policy Three</a>
        </ul>
          <ul class="tab-group">
            <li class="tab active"><a href="#">Fee One</a>
            <li class="tab"><a href="#">Fee Two</a>
            <li class="tab"><a href="#">Fee Three</a>
          </ul>
        </nav>
      </main>
    </body>
  </html
```

4.5 结构化样式

结构化样式包括组件及其容器，比如例 4-12 定义的 `.tabbed-pane` 元素。既然需要为布局定义尺寸，我们可以用结构化样式设置尺寸，然后将其添加给组件和容器。例 4-13 中的代码实现了一种简单的布局，其中包括标题栏、侧边栏和内容区域（图 4-4）。当视口变小时，标题栏、侧边栏和内容区域垂直排列（图 4-5）。

例 4-13 由标题栏、侧边栏和内容区域组成的布局

```
<!doctype html>
<html>
  <head>
    <title>Layout Example</title>
    <style type="text/css">
    *,
    *:after,
    *:before {
      box-sizing: border-box;
    }

    html,
    body,
    main {
      height: 100%;
      margin: 0;
      width: 100%;
    }

    .layout-header {
      background-color: #DDDD88;
      display: block;
      min-height: 10%;
      width: 100%;
    }
```

```
    .layout-header,
    .layout-sidebar,
    .layout-content {
      text-align: center;
    }

    .layout-sidebar,
    .layout-content {
      float: left;
      height: 100%;
    }

    .layout-sidebar {
      background-color: #8888BB;
      width: 20%;
    }
    .layout-content {
      background-color: #EEBB55;
      width: 80%;
    }

    @media all and (max-width: 640px) {
      .layout-header,
      .layout-sidebar,
      .layout-content {
        display: block;
        float: none;
        height: auto;
        min-height: 100px;
        width: 100%;
      }
    }
    </style>
  </head>
  <body>
    <main>
      <header class="layout-header">Header</header>
      <div class="layout-sidebar">Sidebar</div>
      <div class="layout-content">Content</div>
    </main>
  </body>
</html>
```

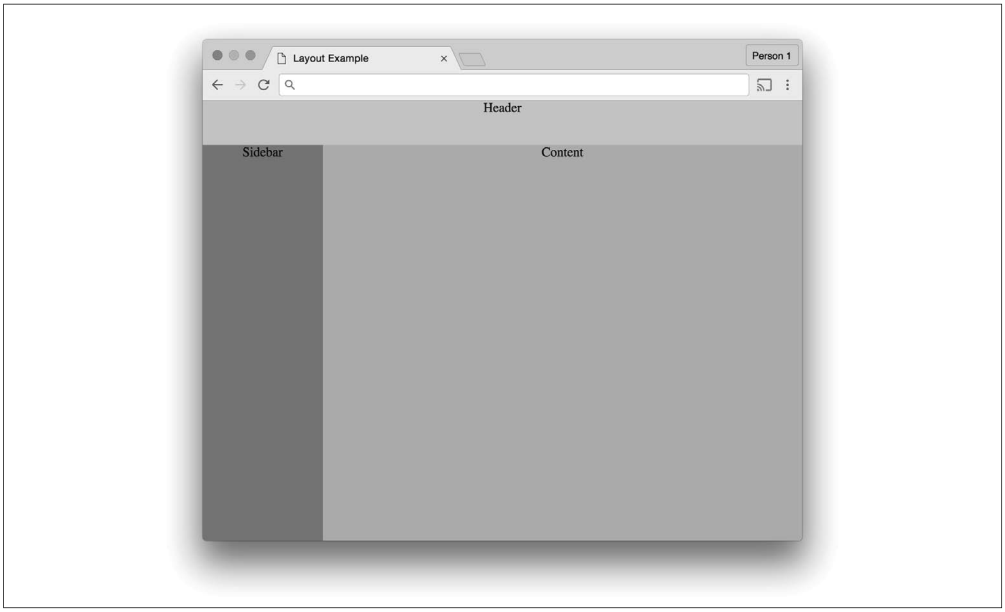


图 4-4：由标题栏、侧边栏和内容区域组成的布局

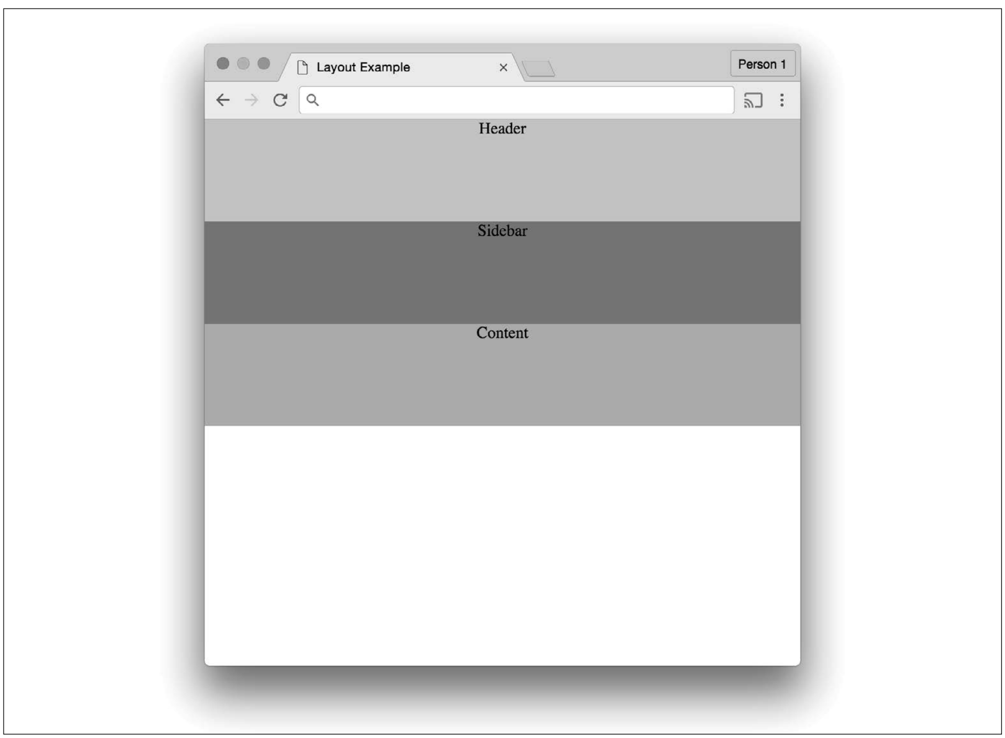


图 4-5：标题栏、侧边栏和内容区域垂直排列的布局

4.6 功能样式

第 2 章讲过，`!important` 声明通过告诉浏览器，某一声明应该用于与其所在的规则集选择器相匹配的元素，而不管声明块的特指度高低，从而改写了级联样式。若误用 `!important` 将导致 CSS 混乱：如果在多个声明块为同一元素定义样式时均使用了 `!important`，规则集的出现顺序将决定应用哪种样式。

不用 `!important` 可以避免 CSS 混乱，但如果合理使用，它的帮助作用会很大。**功能样式**是指谨慎的开发人员在定义 HTML 元素的类时为其指定的样式，或满足特定条件时，用 JavaScript 添加的样式。例如，如果你想按下按钮隐藏某个元素，可以通过添加下面这个类来实现：

```
.hidden {
  display: none !important;
}
```

添加 `!important` 能够确保不管为元素增加或删除其他的类，元素将一直处于隐藏状态（当然，除非在后面再添加 `display` 属性，并用 `!important` 提升优先级引发了冲突）。

4.7 浏览器特定样式

由于老的浏览器有一些怪癖，我们可以使用**浏览器特定 CSS 技术**，引导浏览器按照你的预期工作，以克服这些问题。例如，由于 Internet Explorer 7 的一个 bug，`display` 属性的 `inline-block` 属性值有时无法实现预期效果，但是我们仍然能够让元素表现为 inline-block 元素，方法请见例 4-14。

例 4-14　IE7 的 `inline-block` 问题解决技巧

```
.selector {
  display: inline-block;
  *display: inline;
  zoom: 1;
}
```

上述代码有两个问题。其一，`*display` 属性不合法，因为它存在句法错误（属性名不能以 * 开始）。其二，我们为了兼容过时的浏览器，利用“黑技术”编写了存在句法错误的 CSS。如果你确实无法放弃支持需要采用“黑技术”的浏览器，那么应该将这些语句单独写到一块，并且添加注释来解释这些代码的用途。例如，如果你需要用 CSS 技巧支持老的 Internet Explorer 浏览器，那么应该将其放到单独的样式表中，并用**条件注释**（conditional comments）添加对该样式表的引用，只为特定版本的浏览器加载这些样式。

```
<!--[if IE 7]>
  <link rel="stylesheet" href="ie7.css" type="text/css" />
<![endif]-->
```

4.8 总结

本章介绍了不同用途的样式。虽然这些概念非常简单，却很强大。正如第 6 章要讲的，扎实地理解样式的不同用途以及级联样式的功能，将会使合理调整并复用 CSS 变得更加简单。开始调整和重构 CSS 之前，我们先讨论一下测试。

第 5 章

测试

测试 CSS 有难度，因为不同的平台、屏幕尺寸和设备都需要测试。本章将探讨如何决定需要测试的浏览器和设备，以及测试和维护 CSS 的几种不同方式。学完本章，你应该能够更好地理解如何测试 CSS，带着这些知识重构代码，你将更加自信。

5.1 为什么说测试很困难

全面彻底地对 CSS 代码的改动进行测试会花费较长时间，并且需要用到多种不同工具。

测试时需要考虑很多因素，其中包括以下几点。

- 正在用什么浏览器测试网页？
- 如何在不同的操作系统上测试各种各样的浏览器？
- 正在多大的窗口浏览网页？
- 如何快速测试大量网页？
- 如何验证你所看到的效果是正确的？
- 如果你无法获得某些设备，如何测试网站在这些设备上的效果？

5.2 需要测试的重点浏览器

测试之前，知道哪些浏览器应该被测试是很重要的。理想情况下，你只需要支持多数用户访问网站所用的浏览器即可（具体的阈值因公司而异）。网站用户所使用的浏览器、设备及其版本号，可用网站分析工具获取，非常简单。

5.3 浏览器市场份额

能够支持最近几年发布的所有主要浏览器是很重要的。Chrome、Firefox、Safari 和 Microsoft Edge（以及更早的 Internet Explorer）的最近几个版本及其相应的移动端版本，在兼容浏览器标准方面做了很多了不起的工作，因此它们的行为与过去相比会显得更加一致。以上每种浏览器都提供自动更新功能，这表明随着时间的发展，更多用户都将使用最新的现代浏览器。

然而，不幸的是，一些用户仍旧在使用老式浏览器。例如，写作本书时，根据 NetMarketShare 调查结果显示，Microsoft Edge/Internet Explorer 三个最常用的版本依次为 Internet Explorer 11、Internet Explorer 8 和 Edge 13（图 5-1）。

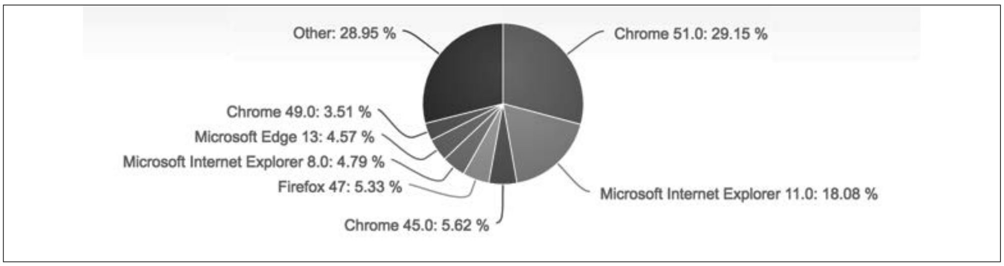
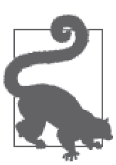


图 5-1：NetMarketShare 公布的 2016 年 7 月桌面浏览器市场份额

如果用户不是频繁地使用老式浏览器访问你的网站，你可能没必要担心维护兼容老式浏览器的代码。然而，如果老式浏览器贡献了可观的流量，那么你也许要考虑继续支持它们，网站若能为你带来收入，则更应如此。

Google Analytics的浏览器统计数据和屏幕分辨率

Google Analytics（http://www.google.com/analytics）是 Google 提供的免费增值服务，它是最常用的网站分析工具集之一。Google Analytics 跟踪网站流量、用户行为等数据。我们一起来看看其中两个对 CSS 重构非常重要的数据：浏览器信息和屏幕分辨率。



在其他分析工具中，查找浏览器统计数据和屏幕分辨率

如果不使用 Google Analytics 而是用其他分析工具，你也许仍然能够找到用户访问网站所用的浏览器和屏幕分辨率数据。请参考分析工具的文档，了解更多信息。

1. 浏览器信息

写作本书时，我选中了“Browser”作为首要查看维度，浏览器信息可从 Google Analytics 的 Audience → Technology → Browser & OS 菜单中找到（图 5-2）。点击“Browser & OS”

菜单，在右侧打开的页面再点击一款浏览器名称，即可查看该浏览器的各个版本访问网站的统计数据，如图 5-3 所示。从这些统计信息中，我们可以找出多数用户使用什么浏览器。如果你发现自己花费大量时间编写的 CSS 支持很少用户使用的老式浏览器，那么应该考虑放弃这种支持。

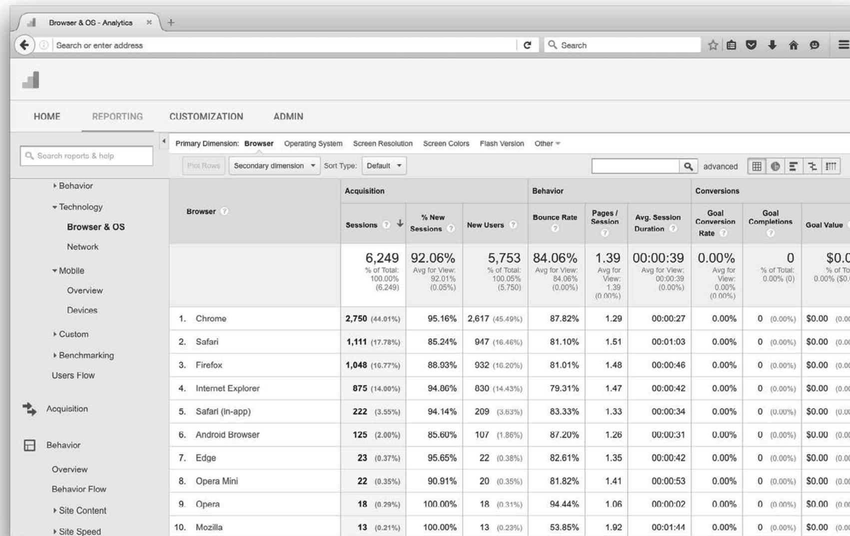


图 5-2：Google Analytics 工具的 Browser & OS 界面

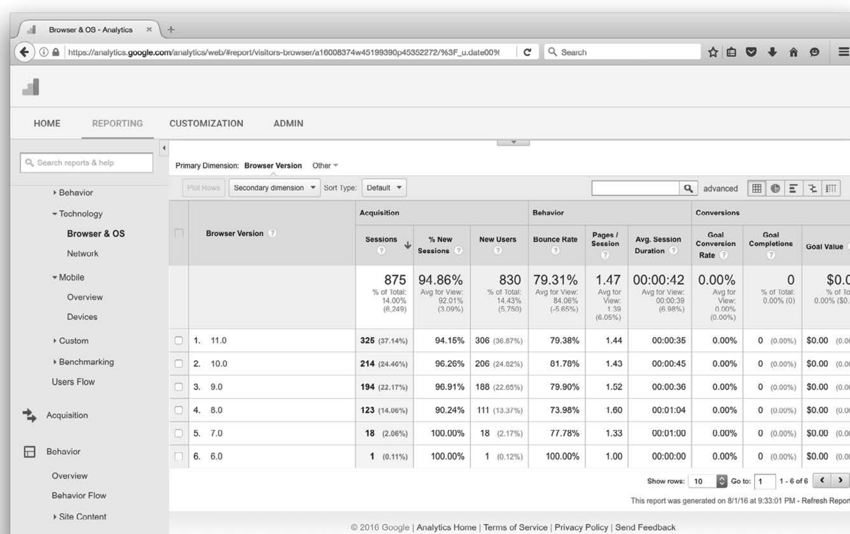


图 5-3：Google Analytics 工具统计的浏览器版本

2.屏幕分辨率

选中“Screen Resolution”作为 Google Analytics 的首要查看维度，然后可从 Audience → Technology → Browser & OS 菜单中找到屏幕分辨率信息（图 5-4）。从这些信息可知用户访问网站时所用设备的屏幕尺寸，并可帮助你确定网站设计所要满足的最普遍的应用场景。

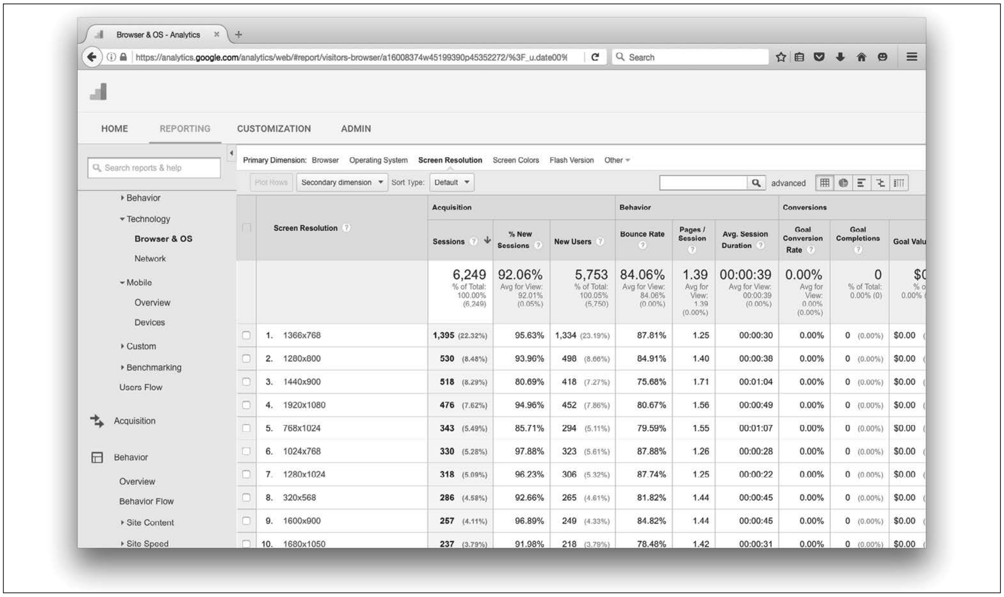


图 5-4：Google Analytics 工具统计的屏幕分辨率

5.4 测试多个浏览器

最常用的测试 CSS 在不同浏览器中显示效果的方法是人工测试。你可能想下载以下所有主流浏览器：

- Google Chrome（http://www.google.com/chrome）
- Firefox（https://www.mozilla.org/firefox）
- Safari（https://www.apple.com/safari）
- Microsoft Edge（https://www.microsoft.com/en-us/windows/microsoft-edge）

为了测试 CSS 在移动端的效果，你需要从合适的应用市场下载适合于设备的各种浏览器。

5.4.1 iOS系统的Safari浏览器

要用 iOS 系统的 Safari 浏览器测试，可以使用 iOS 设备的原生应用或 Xcode 的 iOS 模拟器。你可以从 Apple App Store 免费下载 Xcode（图 5-5），但不幸的是，Xcode 只能在 Mac OS 系统上运行，无法安装到 Windows 系统。



图 5-5：App Store Xcode 应用主页

安装好 Xcode 之后，先从 Xcode → Open Developer Tool → iOS Simulator 启动 iOS 模拟器，然后可以启动 Safari 浏览器，查看网站的显示效果（图 5-6）。



图 5-6：使用 iOS 模拟器中的 Safari 浏览器

5.4.2 安卓

安卓设备可以用 Android Studio 的模拟器测试（图 5-7），Android Studio 可以从 http://developer.android.com 免费下载。

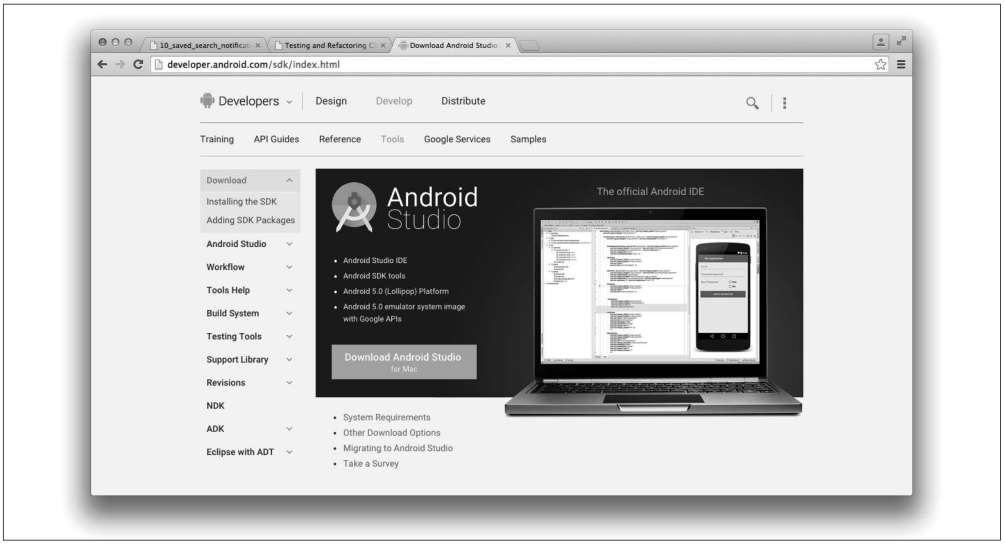


图 5-7：Android Studio 下载页

安装好 Android Studio 之后，新建一个空白项目，然后从 Tools → Android → AVD Manager 菜单选择一种模拟器，完成创建和启动模拟器（图 5-8）。

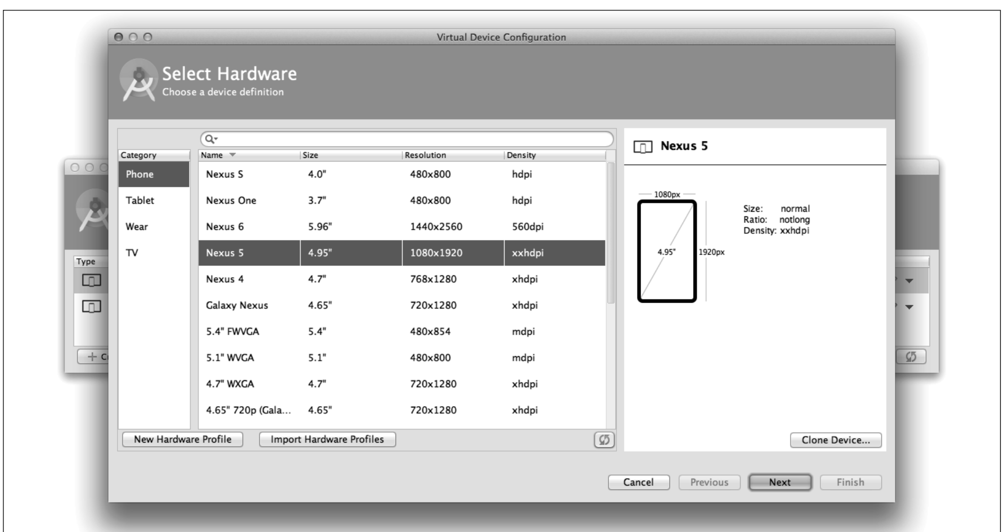


图 5-8：Android 虚拟设备管理器

启动模拟器后，可以在模拟设备中启动 Web 浏览器，并用其查看网站，Android Studio 模拟器非常类似于 iOS 模拟器（图 5-9）。



图 5-9：模拟安卓设备

5.5　测试老式浏览器

本章开头，我们学习了如何用分析工具判断用户是否频繁用过时的浏览器访问网站。有些情况下，你也许会放弃支持老式浏览器——但是如果无法放弃，你需要确保这部分用户依旧能够正常使用网站。下面几节介绍如何测试网站在老式浏览器中的显示效果。

5.5.1　Internet Explorer和Microsoft Edge

你可以从 https://www.modern.ie 下载 Microsoft Edge 和老式的 Internet Explorer 用于测试。这些虚拟机形式的浏览器，可以运行在 VirtualBox（https://www.virtualbox.org/）、VMWare（http://www.vmware.com）或 Parallels（http://www.parallels.com）虚拟机软件之中。**虚拟机**是一种模拟操作系统的软件，VirtualBox、VMWare 和 Parallels 是可以运行虚拟机的程序。虚拟机的用处很大，Internet Explorer 等软件的编译方式决定了它们只能运行在特定的操作系统上，但是虚拟机能够提供浏览器可以正常运行的虚拟环境。

5.5.2　Firefox浏览器

老式 Firefox 浏览器下载地址为 https://support.mozilla.org/en-US/kb/install-older-version-of-firefox。测试网站在老式 Firefox 中的效果很简单，只要找到你需要的浏览器版本，根据操作系统型号下载，然后查看网站在浏览器中的效果即可。如果你不是在 Windows 系统中开发，但是使用了 Internet Explorer 虚拟机，那么也可以在虚拟机中下载兼容 Windows 系统的老式 Firefox 测试网站。

5.5.3　Safari和iOS系统的Safari

很不幸，测试旧版本的 Safari 浏览器需要旧版本的 Mac OS，因为 Safari 使用 OS X 中的 WebKit 框架渲染网页。然而，测试 iOS 系统的旧版本 Safari 浏览器可以用 Xcode 模拟器依次点击 Hardware → Device → Manage Devices 菜单。在 Devices 窗口，选择目标操作系统，创建一个新的模拟器。（图 5-10）。

关于 iOS 模拟器的更多信息，请见 Simulator User Guide（模拟器用户指南）（http://apple.co/2fgPFNv）。

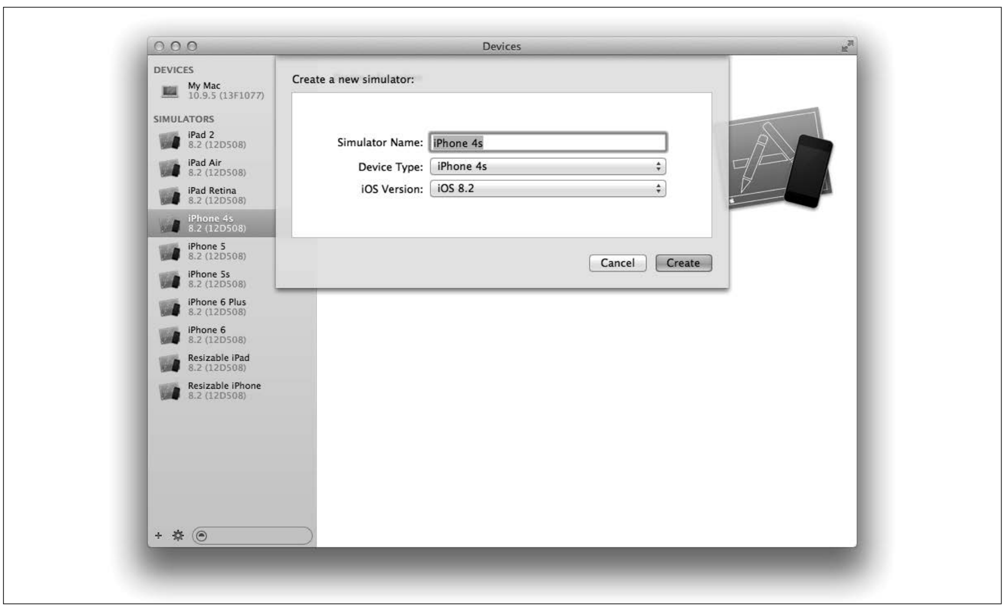


图 5-10：在 iOS 模拟器中新建模拟器

5.5.4　Chrome浏览器

不幸的是，Google 不提供旧版本的 Chrome 浏览器用于测试，但是每六周会发布一个新版本，并且每个新版本的采用率都非常高。

5.6 测试最新版本的浏览器

如果你对 Chrome、Firefox 或 Safari 的下一个版本感兴趣，可以从以下网站及时了解它们前一天晚上刚发布的最新版本：

- Chrome Canary（https://www.google.com/chrome/browser/canary.html）
- Firefox Aurora（https://nightly.mozilla.org）
- WebKit Nightly（http://nigthly.webkit.org）

Microsoft 更新 Edge 的频率没有这么高，但是确实经常为 Windows Insider Program（https://insider.windows.com）成员提供新版浏览器的早期版本。

5.7 第三方测试服务

自己执行测试任务，要维护多个操作系统、大量浏览器和模拟器。除了自己测试，你还可以使用第三方提供的种类丰富的测试服务。它们能够满足你测试网站在任意浏览器或其他配置环境中的效果的需求，同时还提供类似共享测试阶段信息、人工测试多种浏览器和截图等功能。下面列出的几种第三方服务，有些可免费试用一段时间或提供几种级别的免费服务：

- BrowserStack（https://www.browserstack.com）
- Sauce Labs（https://saucelabs.com）
- Browserling（https://www.browserling.com）
- Litmus（https://litmus.com）

5.8 用开发者工具测试

主流浏览器自带开发者工具，可以帮助开发者打造更好的网站：每种工具集提供多种工具，每种工具可用来完成不同任务，但是本节只讨论与 CSS 测试相关的工具。

- Chrome DevTools（https://developer.chrome.com/devtools）
- Safari for Developers（https://developer.apple.com/safari/tools/）
- Firefox Developer Tools（https://developer.mozilla.org/en-US/docs/Tools）
- Microsoft Edge Developer Tools（https://dev.modern.ie/platform/documentation/f12-devtools-guide/）

不断发展的开发者工具

写作本书时，下面列出的开发者工具指南准确无误，但是浏览器开发者工具在不断改进。如果你发现不可避免的改进使得下面这些信息已经过时，请参考前面的浏览器开发者工具集网站，了解更多信息。

5.8.1 模拟设备尺寸

要测试网站在多种设备（例如：手机、平板等）上的显示效果，当然可以购买大量设备，但是昂贵的费用很快就会令人望而却步。另一种可行的方法是用浏览器开发者工具模拟设备的尺寸。我们可以通过调整浏览器窗口模拟设备尺寸，但是使用预先设置好的尺寸会更快捷。Google Chrome 提供多种预先定义好的尺寸，使用时可从 DevTools 的 Device Mode（设备模式）菜单查找。

可以通过以下方式进入 Device Mode 菜单：

- 右键点击网页的任意位置
- 选择“Inspect Element（查看元素）”，打开 Chrome DevTools
- 点击 Toggle Device Mode（切换设备模式）图标（图 5-11）



图 5-11：Chrome 的 Toggle Device Mode 图标

在 Device Mode 模式下，窗口顶部会出现工具栏，用以提供预先设置好的设备尺寸，你可以根据需要进行设置（图 5-12）。

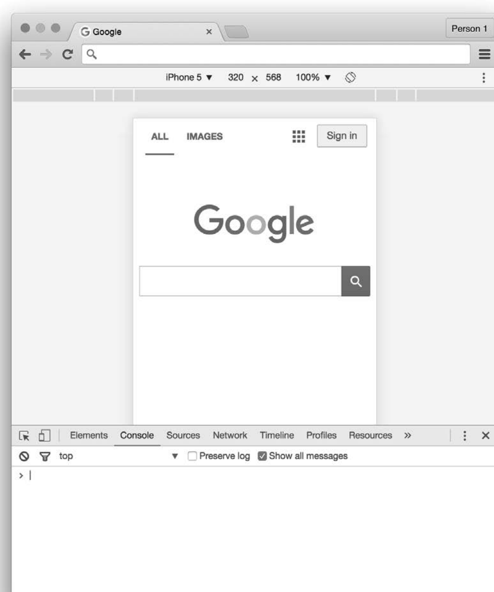


图 5-12：Chrome DevTools 的设备模拟器

从 Firefox 的 Tools → Web Developer 菜单，进入 Responsive Design（响应式设计）模式，可在 Firefox 浏览器中模拟出不同设备尺寸（图 5-13）。

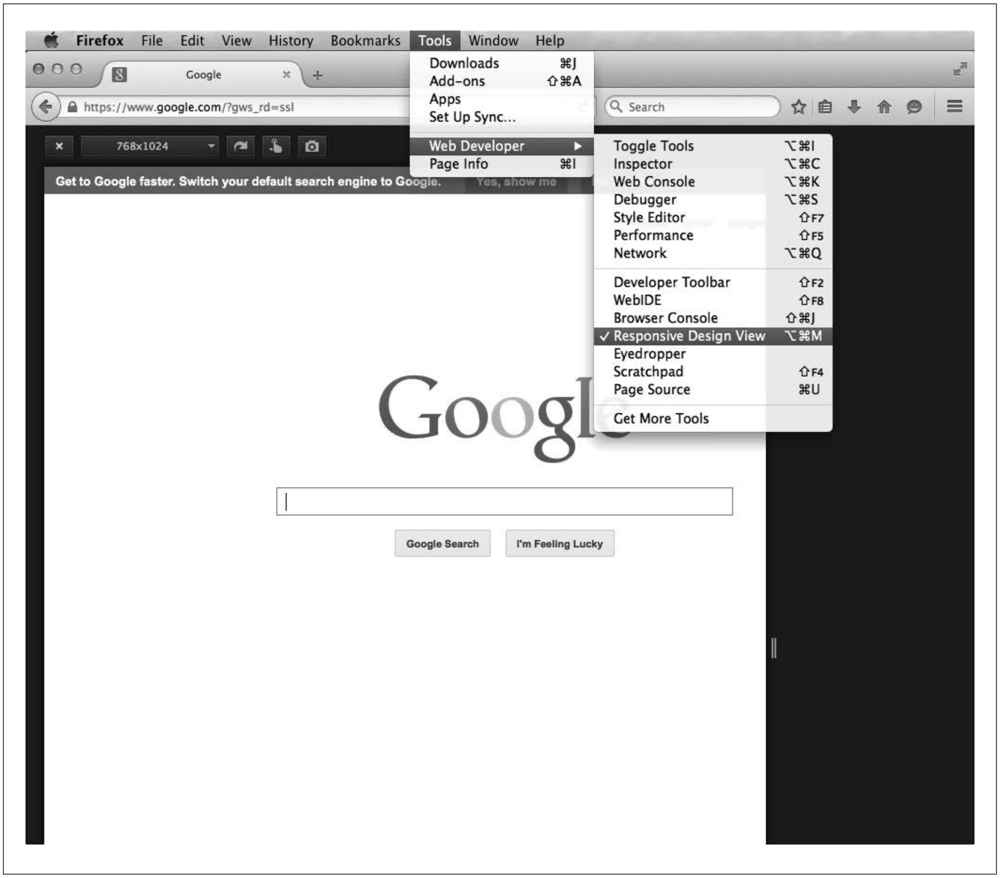


图 5-13：Firefox 的响应式设计模式

Safari 浏览器也有响应式设计模式，可以模拟不同设备尺寸（图 5-14）。从 Develop → Enter Responsive Design Mode 菜单项可以进入该模式。



图 5-14：Safari 的响应式设计模式

如要在 Microsoft Edge 浏览器模拟不同设备尺寸，请参照以下步骤。

(1) 按下 F12，打开开发者工具。
(2) 点击 Emulation（模拟）页卡（图 5-15）。
(3) 从“Resolution（分辨率）”下拉菜单选择设备尺寸。

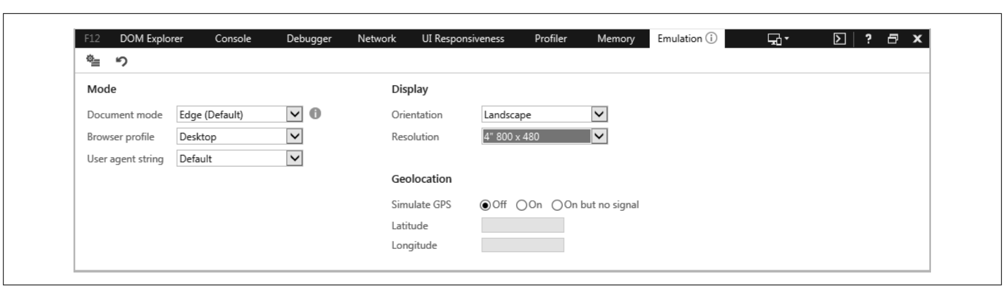


图 5-15：Internet Explorer 的 F12 开发者工具 Emulation 页卡



用户代理仿造

用户代理是 Web 浏览器向服务器发送的，用以标识自己身份的字符串。所有主流浏览器提供的开发者工具都支持修改用户代理，并且人们往往认为修改该值，其效用等价于模拟另外一种浏览器。然而，仿造用户代理只改变浏览器识别自己的方式；修改该字符串无法改变浏览器的渲染引擎。

5.8.2 文档对象模型（DOM）和CSS样式

如今的开发者工具支持以交互式方式访问 DOM，支持查看元素，或通过增加、删除、修改属性或样式的方式操纵元素。从 DOM 中选中元素，可分析元素应用的样式和属性的计算结果。

图 5-16 展示的是为某一网站的 `<body>` 元素添加样式所使用的 CSS 规则，其样式的来源有多种。通过继承得到的规则，清楚地标明了出处及其在源文件中的行号。伪元素 `::before` 和 `::after` 的样式也列了出来。标有删除线的规则是指按照级联方法被其他规则覆盖的样式。不同的浏览器在开发者工具中添加或操纵样式的方式有所不同。

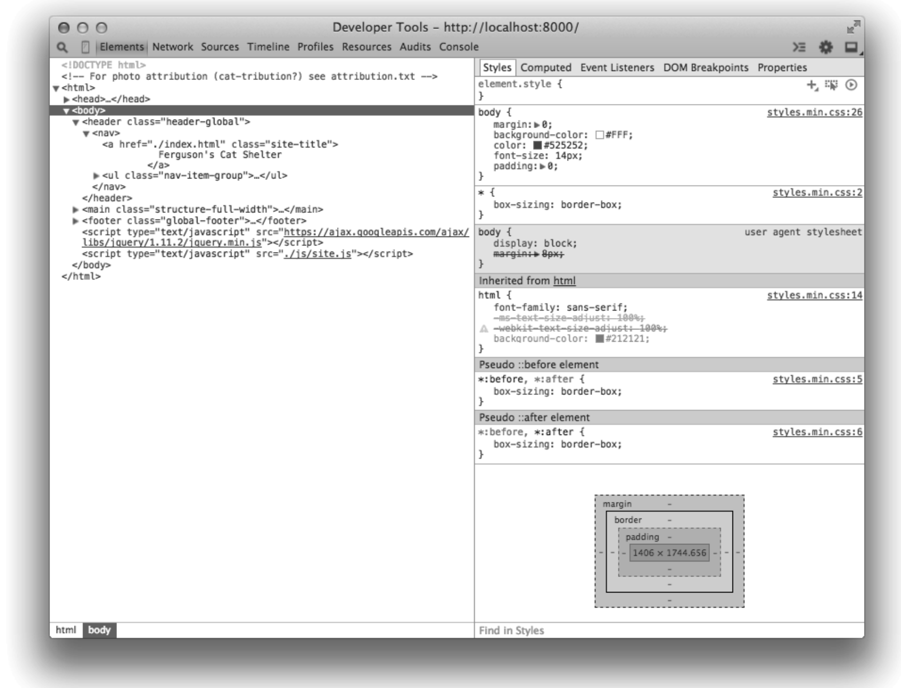


图 5-16：Chrome DevTools Styles（样式）面板

我们还可以查看元素应用级联样式后的显示效果，如图 5-17 所示。对于经过扩展得到的样式，开发者工具将显示样式的源文件及其行号。

在 DOM 中查看元素，右击该元素（Mac OS X 系统，按下 Ctrl 后单击），从 context 菜单选择“Inspect Element（查看元素）”。浏览器的开发者工具将自动启动，在 DOM 查看器中高亮显示所选元素。在 Chrome、Firefox 和 Edge 浏览器中，Styles 面板默认位置在 DOM 查看器的一侧。在 Safari 浏览器中，你也许需要单击开发者工具菜单右上角的 Styles 页卡以使用该工具。

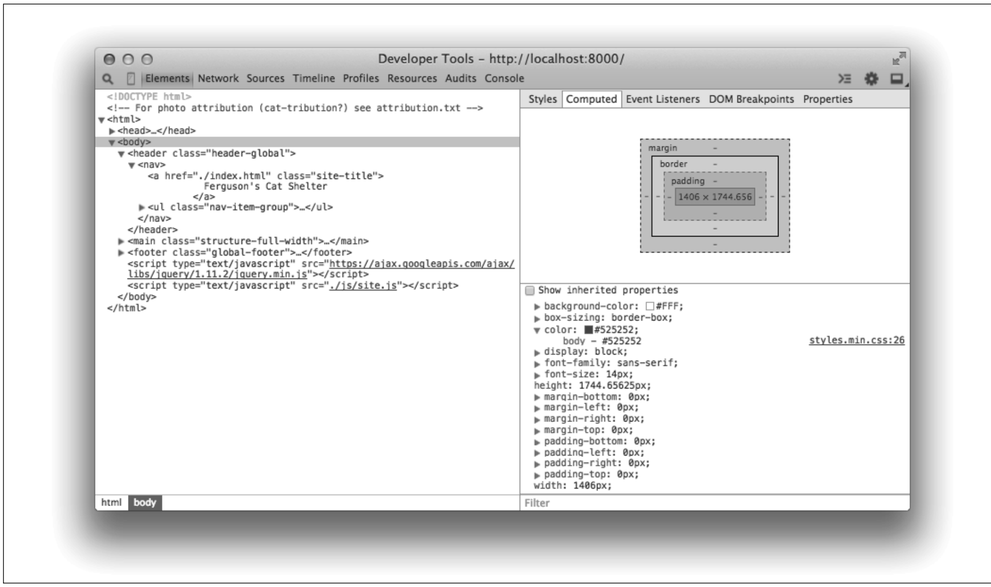


图 5-17：Chrome DevTools 计算样式页卡

5.9　视觉回归测试

视觉回归测试是一种测试方法，它通过比较作为基准的用户界面图像和开发过程同一用户界面的图像，来检测不符合预期的改动（回归[1]）。视觉回归测试非常耗时，因为需要测试的浏览器很多，一有改动就进行这种测试，测试工作量很大。此外，肉眼难以确定元素在空间位置上的变化。

虽然视觉回归测试帮助作用很大，但是跟其他任何一种测试方法类似，它并不是包治百病的灵丹妙药。视觉回归测试是用来识别已经出现的错误，因此在将 CSS 改动提交到生产环境之前，进行该类测试很有必要。

注 1：视觉效果上的变动，多指负面。——译者注

5.9.1　视觉回归测试技巧

视觉回归测试有如下技巧。

测试重要的点

　　类似于单元测试，在进行视觉回归测试时，你可能想测试自己编写的所有代码。然而，测试的代码越多，你要维护的就越多，而且大量测试不一定就是高质量的测试。与其进行大量测试，不如只测试真正需要测试的点。例如，一旦基础样式定义好，就不大可能因为改动它们而引入错误；但是更为复杂、更加脆弱的可用性组件则是需要重点测试的。

保持测试的粒度

　　执行视觉回归测试时将整个网页截图可一次性测试多个组件，但是你应该避免这样做。同时测试大量组件，难以决定引起回归的真正原因，因为有很多不同的事情在发生。因此，测试要保持足够细致的粒度，每次只测单个组件。如果产生了回归现象，那么寻找起因也更加容易。

使用多种浏览器

　　用多种浏览器进行视觉回归测试极其重要，因为不同浏览器之间可能存在不一致现象。也就是说，不要尝试比较不同浏览器的截图，因为该做法效率不高。如果一种浏览器渲染文本的方式与另一种浏览器存在细微差异，很可能不算是什么大问题，除非它确实破坏了页面展示效果。这还表明，如果你需要测试 Internet Explorer 或 Safari 的不同版本，你可能需要准备多种测试环境，方法本章前面已经讲过。

5.9.2　用Gemini执行视觉回归测试

Gemini（https://github.com/gemini-testing/gemini）项目是 Yandex 团队（https://www.yandex.com）开发的视觉回归测试工具。使用该工具，你可以编写脚本，自动截取网站在主流浏览器中的截图，然后将其与基准图像比较，不同之处将以高亮形式标记出来。

我们将通过下面的示例来学习如何用 Gemini 测试由 WebKit 渲染引擎渲染的界面。我们设置好让 Gemini 用无头浏览器 PhantomJS（http://phantomjs.org）访问网页。**无头浏览器**是指没有用户界面，为其他程序提供内容的 Web 浏览器。它不仅能够在不展示网页的情况下渲染页面、截图，还常用来为开发人员提供与浏览器交互的接口，使其能够向浏览器传递指令。

> **安装 Gemini**
>
> 安装 Gemini 之前，要先安装 Node.js 和 NPM。你可以从 https://nodejs.org 下载 Node.js。Gemini 安装指南请见 https://github.com/gemini-testing/gemini。下面示例需要安装 PhantomJS。你可以从 http://phantomjs.org 下载适合自己操作系统的 PhantomJS 二进制安装包。
>
> Gemini 通过利用 Selenium 或 Sauce Labs 和 BrowserStack 之类的云端服务，还可以测试其他浏览器。详细方法请参考相关文档。

1. 配置

安装 Gemini 之后，你需要在自己项目的根目录中创建名为 .gemini.js 的文件。该示例非常简单，按照例 5-1 配置即可。完整的配置项，请见 https://github.com/gemini-testing/gemini。

例 5-1　Gemini 配置文件

```
module.exports = {
    rootUrl: 'http://127.0.0.1:8000',
    browsers: {
        phantomjs: {
            desiredCapabilities: {
                browserName: 'phantomjs'
            }
        }
    }
};
```

上述代码告诉 Gemini，网站的根目录地址为 http://127.0.0.1:8000。设定好根目录地址后，正如下一节要讲到的，我们编写测试代码指定 URL 时，可使用相对于根目录地址的相对路径。上述代码还告诉 Gemini，我们将使用 PhantomJS 浏览器截取图像，并将其保存到名为 phantomjs 的文件夹中。若用多个浏览器进行测试，将有助于区分屏幕截图来自哪个浏览器。

2. 测试

接下来，你需要编写测试文件。Gemini 有自己的一套函数，这些函数能够完成打开 URL、选择特定元素、操纵窗口和截图等操作。如要深入了解这些功能，请阅读其文档。该例中，我们将使用例 5-2 所示的代码。

例 5-2　Gemini 测试代码文件

```
gemini.suite('animals', function(suite) {
    suite.setUrl('/')
        .setCaptureElements('.animal')
        .capture('plain')
});
```

上述代码相当直接。首先，声明一个测试任务，并为其起一个名字（该例使用“animals”）。然后，定义一个要执行的函数，函数体为测试步骤。用 `setUrl` 函数指定网站的首页（`/`）为要打开的页面。接着，用 `.setCaptureElements` 函数选择要测试的元素。最后，用 `.capture` 函数截图。

3. 采集基准图像

测试之前，我们需要采集基准图像，Gemini 将对比新截取的图像和基准图像之间的差别。要采集基准图像，需要在终端运行 PhantomJS 和 Gemini。

打开一个终端窗口，输入命令 `phantomjs--webdriver=4444`。参数 `--webdriver=4444` 将在端口 4444 以 WebDriver 模式运行 PhantomJS。Gemini 使用该端口与浏览器进行交互。

接着，另起一个终端窗口，进入到 .gemini.js 文件所在目录，执行命令 `gemini update tests/gemini/animal-tests.js`，采集基准图像（animal-tests.js 为我们刚才编写的测试文件，它位于 tests/gemini 目录下）。采集完基准图像，将其存储于新目录 gemini/screens/animals/plain 下的 phantomjs.png 文件中。该例只采集一个基准图像文件，如图 5-18 所示。

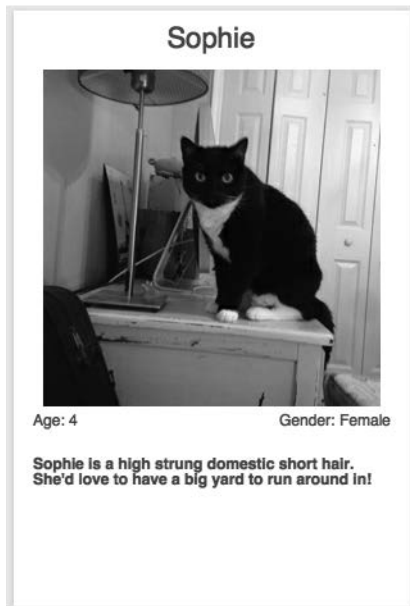


图 5-18：基准图像

4.测试回归

采集到基准图像之后，就可以测试回归情况了。因为要测试的代码是一段正确无误的新代码，所以为了讲解测试方法，我们将对代码做简单的改动。改动了我们要测试的 CSS 组件的代码后，用命令 `gemini test --reporter html tests/gemini/animal-tests.js` 运行 Gemini。该命令要求 Gemini 运行测试，比较前后两次的截图，输出对比结果为 HTML 页面，该页面存储在 gemini-report/index.html 中。打开该文件，将看到测试结果（图 5-19）。

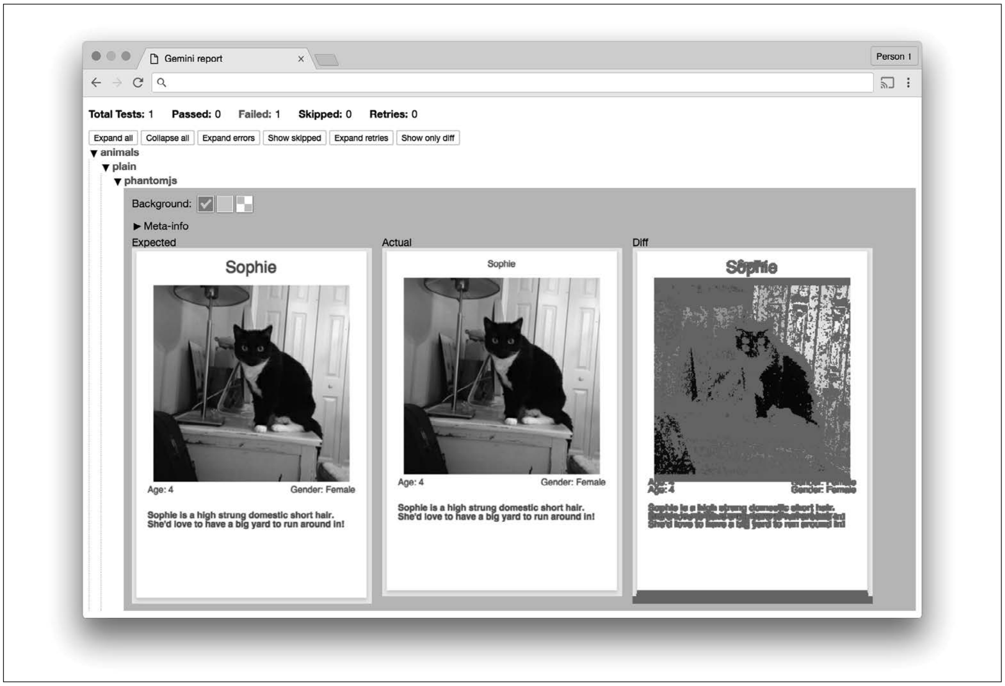


图 5-19：Gemini 报告

Gemini 生成的报告，既是对测试结果的总结，也展示了测试时使用的截图。由图 5-19 可见，两幅图的差别非常大，但是通过分析基准图像和现有图像，我们可以发现实际上是字体方面的问题，引起了组件尺寸的变化。

Gemini 的替代方案

除了 Gemini，还有多种视觉回归测试工具，其中最常用的两个是 Wraith 和 PhantomCSS。这两个测试工具的设计理念非常类似，它们能打开网站，能用 PhantomJS（基于 WebKit 的无头浏览器）或 SlimerJS（基于 Gecko 的无头浏览器）截图，能对比网页现有元素的截图跟元素基准图像之间的差异。

Wraith（http://bbc-news.github.io/wraith/index.html）是由 BBC 开发的。它依赖于 CasperJS、PhantomJS 或 SlimerJS、ImageMagick 和 Ruby。

PhantomCSS（https://github.com/Huddle/PhantomCSS）由 Huddle（http://huddle.com）公司的 James Cryer 带领开发团队编写。它依赖于 CasperJS 和 Resemble.JS，可以结合 PhantomJS 或 SlimerJS 使用。

5.10　维护你的代码

代码的测试跟编写同等重要。不断维护已有代码，提高其质量，其重要性不亚于编写新代码。正如我们在下面几节将要学习到的，遵循代码标准和应用模式库可以保证代码的质量。

5.10.1　编码规范

编码规范是指将良好的代码编写方法记录下来形成指南，以鼓励团队所有成员以相同的方式编写代码。编码规范应该由集体创作而成，并定期审阅和更新。随着技术的发展，编码规范有助于团队跟踪最新的技术，代码评审在这方面所起的作用尤为明显。CSS 编码规范通常指定了注释、格式、命名和选择器用法方面的规范，其详略程度可根据实际情况自行调整。

下面我们来看例 5-3 编码规范示例。你和你的团队可以以此为出发点，来讨论在自己的项目中应该如何编写 CSS。你们可以以适合自己的方式编写和存储编码规范，但是团队的每个成员都应该能够很方便地参考它们。

例 5-3　编码规范示例

```
(1)注释

A．应该在每个文件的开头添加注释，说明文件的内容。

    /**
     * 该文件包含选项卡组的样式。
     * 选项卡组应仅包含拥有tab类的元素。
     */

B．易于混淆的属性，应用注释予以说明。

    .tab-group-flush {
        display: block;
        margin-left: -12px; /* 清除父容器的padding值 */
        margin-right: -12px; /* 清除父容器的padding值 */
    }

(2)格式

A．规则集应该满足下列要求。

    —有多个属性时，每个属性占一行
    —规则集声明块中的每条声明缩进4个空格

    /* 错误 */
    .selector {
    property1: value;
    property2: value;
    }
```

```
/* 错误 */
.selector {
        property1: value;
        property2: value;
}

/* 错误 */
.selector { property1: value; property2: value; }

/* 正确 */
.selector {
    property1: value;
    property2: value;
}
```

B. 声明语句应该满足下列要求。

—冒号后面加1个空格
—必须以分号结尾

```
/* 错误 */
.selector {
    property1:value;
}

/* 错误 */
.selector {
    property1: value
}

/* 错误 */
.selector {
    property1 : value;
}

/* 正确 */
.selector {
    property1: value;
}
```

C. background-position各个属性值不同时，可以将两个属性值放在一行。

```
/* 错误 */
.selector1 { property1: value; property2: value; }
.selector2 { property1: value; property2: value; }
.selector3 { property1: value; property2: value; }

/* 正确 */
.selector1 { background-position: 0 0;     }
.selector2 { background-position: 0 -10px; }
.selector3 { background-position: 0 -10px; }
```

D. 规则集和声明末尾的空格必须删除。

(3) 选择器命名规范

A. 只允许使用小写字母。

```
/* 错误 */
.SeleCtor {}
.SELECTOR {}

/* 正确 */
.selector {}
```

B. 包含多个单词的选择器必须使用脊柱状形式（用连字符连接单词）。

```
/* 错误 */
.selectorWithMultipleWords {}
.SELECTORWITHMULTIPLEWORDS {}
.selector_with_multiple_words {}
.selectorWith_multiple-words {}

/* 正确 */
.selector-with-multiple-words {}
```

C. 禁止用ID为元素添加样式，应该使用类。

```
/* 错误 */
#element-to-style {}

/* 正确 */
.element-to-style {}
```

D. 用JavaScript修改样式（不管用什么框架），都必须通过增加或删除CSS类来完成。

```
/**
 * 错误：在JavaScript中，操作元素的样式属性修改了元素的样式。
 */
$('.js-menu-item').on('click', function (e) {
    $(this).css('background-color', '#FFFF00');
});

/**
 * 正确：用JavaScript为元素添加类，修改元素的样式。
 */
$('.js-menu-item').on('click', function (e) {
    $(this).addClass('highlighted');
});
```

E. 用作JavaScript选择器的类和ID，必须添加js-前缀，并严禁在样式表中使用。

```
/**
 * 错误：带有js-前缀的样式严禁在样式表中使用。
 */
#js-element-only-used-by-javascript {
    background-color: #FFFF00;
}
```

```
/**
 * 错误：在JavaScript中，用为元素添加样式的类选择元素。
 */
$('.menu-item').on('click', function () {
    $(this).addClass('highlighted');
});

/**
 * 正确：在JavaScript中，用专门用作JavaScript选择器的类选择元素。
 */
$('.js-menu-item').on('click', function () {
    $(this).addClass('highlighted');
});
```

F. 必须使用有意义的类名。

```
/* 错误：类的命名无意义。 */
.r {}

/* 正确：类的命名有意义且描述清楚。 */
.resident {}
```

G. 类名必须描述为什么元素添加样式，而不是怎样添加样式。

```
/* 错误：类的命名描述的是添加的样式。 */
.float-left-bold {}

/* 正确：类的命名描述的是为什么元素添加样式。 */
.sidebar-important {}
```

(4) 属性

A. 属性的简写形式只可用于border、margin和padding。

```
/* 错误：font属性使用了简写形式。 */
.selector {
    border: 1px solid #000000;
    font: 12px Arial, sans-serif;
}

/* 正确：仅border属性使用了简写形式。 */
.selector {
    border: 1px solid #000000;
    font-family: Arial, sans-serif;
    font-size: 12px;
}
```

B. 属性必须按照字母顺序排列。

```
/* 错误 */
.selector {
    padding: 12px;
```

```
    margin: 24px;
    border: 1px solid #000000;
}

/* 正确 */
.selector {
    border: 1px solid #000000;
    margin: 24px;
    padding: 12px;
}
```

c. 属性值为0时，必须省略单位。

```
/* 错误 */
.selector {
    padding: 0px;
    margin: 0px;
    border: 1px solid #000000;
}

/* 正确 */
.selector {
    border: 1px solid #000000;
    margin: 0;
    padding: 0;
}
```

更多编码规范方面的启示，请见下面这些规范：

- Google CSS 编码规范（http://bit.ly/2e68N3y）
- WordPress CSS 编码规范（http://bit.ly/2fgLrp2）
- 18F 前端指南（https://pages.18f.gov/frontend/#css）

5.10.2　模式库

模式库（有时也称样式指南）是网站使用的一组用户界面模式，它展示了每种模式相关的重要信息，其中包括以下几点：

- 何时（不）使用模式的指导
- 解释模式使用方式的示例代码
- 使用某一模式而不用另一模式的原因

图 5-20 是 Yelp（https://www.yelp.com/styleguide）所使用的模式库。

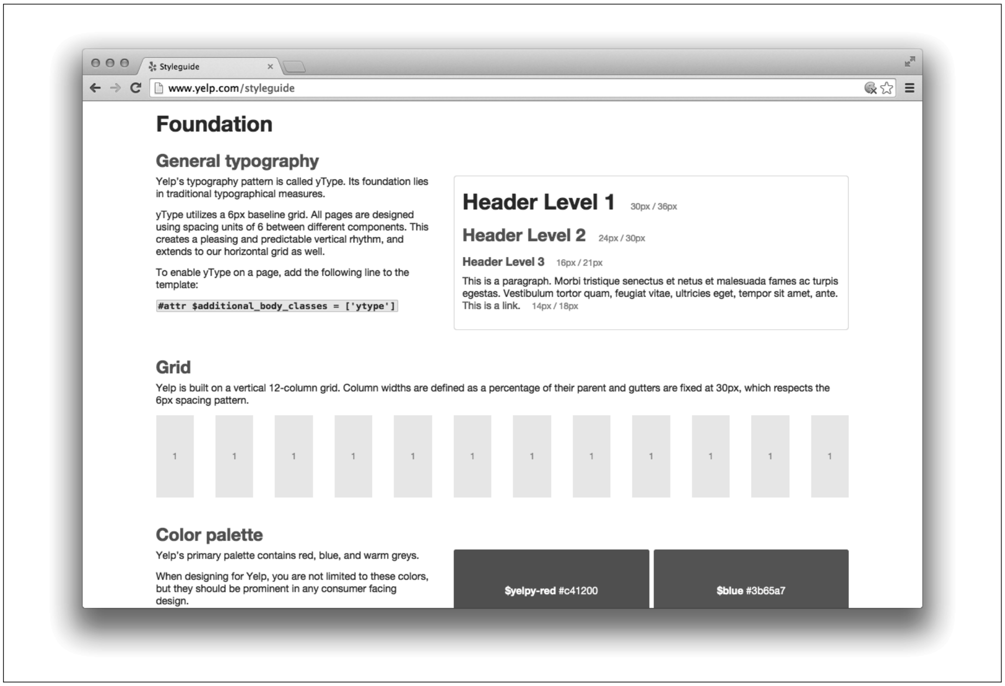


图 5-20：Yelp 模式库

1. 优点

模式库有如下几个优点。首先，模式库将网站所有组件汇集到一起。参与项目的所有成员都能了解到搭建网站的各个模块，确保他们熟悉其背后的原理。让每个人都熟悉一组可复用的模式，还能加快开发速度，因为开发新项目时，无需从头重新开发这些构建模块。

模式库将所有组件汇集到一起，还有助于保证用户界面的一致性。使用模式库对设计团队也能起到帮助作用，因为需要修改组件的样式时，在现有模式的基础上做修改即可。模式库为设计工作提供约束条件，鼓励设计师在现有模式的基础上设计新的元素，这进一步强化了用户界面应该保持一致的设计理念。

最后，模式库将网站的所有组件都汇集到一起，使识别不一致的组件变得更加容易。编写的新代码可能影响到用户界面效果，在提交代码之前，借助模式库，可从视觉上快速识别错误。修改模式后，若网页看起来有问题，有了模式库，诊断问题将更加容易，因为模式库是模式的最简单应用形式。

2. 建设模式库

模式库不需要建成艺术品库，它只需要展示每种模式及其相关信息。模式的繁简视需要而定。模式库应放到团队成员都能访问到的公共位置，以便大家经常参考。

模式库应该是团队成员共同努力的结果，因为大家都参与进来，可以鼓励更多的人熟悉网站所用的视觉设计模式。实现模式库的过程，也是为团队成员提供对话的机会，大家可以就用户界面到实现的一切内容发表意见，由此还可获得大量新的认识。如果你们还没有模式库，应该先建设一个记录当前用户界面情况的模式库，然后再在此基础上迭代。

模式库发扬了样式应该有不同的用途这一理念，它很好地展示了如何像组合构建模块一样组合样式以实现更复杂的模式。理想情况下，模式库中的每种模式，线上什么地方用到了，要给出其中一处参考位置，此外还要记下每种模式的使用场景以及什么情况下不宜使用，并附上实现该模式的示例代码。模式库还是开始执行视觉回归测试的好地方。MailChimp 的模式库（https://ux.mailchimp.com/patterns）（图 5-21）在上述各个方面都做得非常出色。

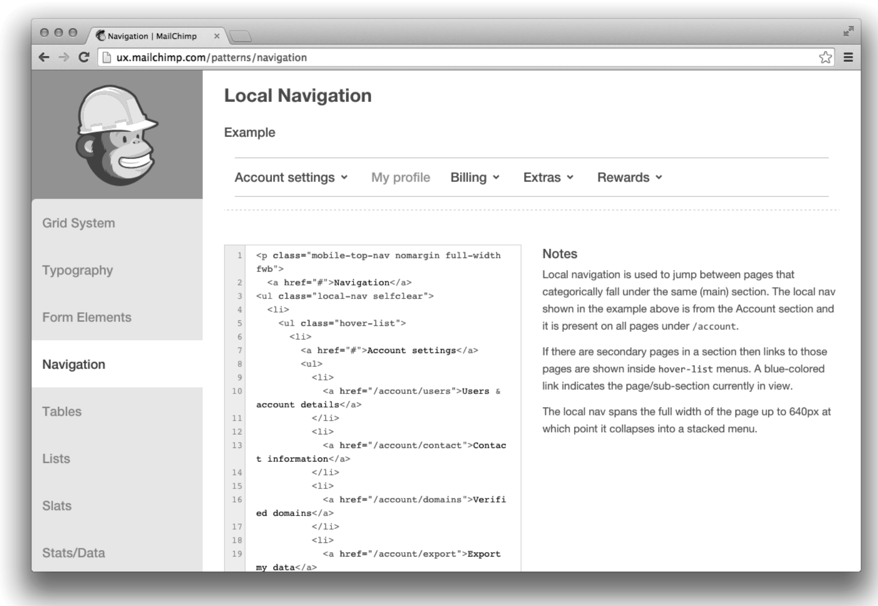


图 5-21：MailChimp 的模式库

更多资源

关于模式库的更多资源，请见 http://styleguides.io，该网站提供相关示例、文章、图书和播客。

5.11 总结

这一章我们学习了 CSS 测试的相关内容。首先，本章探讨了为什么在改动 CSS 后对其测试既费时、难度又大，这是因为有很多浏览器和平台需要测试。然后，本章介绍了各种新旧浏览器、模拟器、虚拟机和第三方平台提供的测试服务的获取方式。接着，本章又讨论了建设编码规范和模式库的优点，介绍了如何用 Gemini 工具执行视觉回归测试。熟练掌握这些工具，CSS 测试将更易于操作。

第 6 章

代码的组织和重构策略

前几章我们学习了大量 CSS 的相关知识，包括级联方法的工作原理，样式的不同用途，编写更优质的 CSS 以及提高测试效率的方法。本章将首先讨论样式的组织方式，然后探索如何利用我们所学的一切知识更好地重构 CSS，最后再探索如何评价代码重构是否成功。

6.1 按照样式从最不精确到最精确组织CSS

我们从第 2 章学习到，CSS 样式根据选择器的特指度和样式的顺序产生作用。因此，有必要按照样式产生作用的顺序组织 CSS 代码。

(1) 通用样式
(2) 基础样式
(3) 组件及其容器的样式
(4) 结构化样式
(5) 功能性样式
(6) 浏览器特定样式（如果一定需要）

按照以上顺序添加 CSS，随着声明块选择器的精确度提高，更为复杂的选择器将与已经添加的更加宽泛的选择器区分开来。

下面我们来复习不同类型样式的用途。阅读这些样式的用途时，请留意后者是如何在前者的基础上进行定义的。

6.1.1　通用样式

第 4 章讲过，通用样式用来设定基准，以消除不同浏览器之间的不一致性。消除了不一致性之后，再定义样式时就可以使用通用样式定义的属性了。

6.1.2　基础样式

基础样式为网站的所有元素提供基本的样式。留白（`margin`、`padding` 和 `line-height` 等）、字体及其大小等大多数元素都要用到的样式，不需要不断重写，应该在基础样式中进行定义。

6.1.3　组件及其容器的样式

组件样式是以基础样式为基础进行定义的，它利用视觉隐喻，使得用户更易于与网站交互。前面已讲过，该类样式应该能够满足网站范围内的大多数应用场景，样式上的任何调整都应该交由父容器处理。

6.1.4　结构化样式

结构化样式包括组件及其容器的样式。该类样式用来创建网页的布局，并常用于定义尺寸。

6.1.5　功能性样式

功能性样式是所有样式中最精确的样式。JavaScript 所使用的添加 `!important` 语句的类就属于这类样式，其他为满足单一目的而实现的样式也属于该类。

6.1.6　浏览器特定样式

最后，如果你无法放弃支持老式浏览器，那么你专门为其实现的样式属于该类。该类样式可能也使用了 `!important` 语句，只对特定的浏览器生效。它们通常不够优雅，因此不再使用时，一定要将其删除。

媒体查询要靠近相关声明块

媒体查询用于根据条件为元素应用不同的样式，比如根据浏览器视口的宽度变化调整样式。应该将媒体查询写到它们所作用的声明块附近，而不是将其置于 CSS 文件的末尾或单独的文件中，因为这样做可以为样式是如何起作用的提供更多背景信息。

6.2 多个文件还是一个大文件

代码的组织方式有两种，即将代码置于多个文件或只用一个大文件。代码放置位置要易于开发人员查找，这一点很重要，并且同样非常重要的是，这样做有助于网站快速加载以满足终端用户的需要。我们首先一起看看将 CSS 传给浏览器时会发生什么，然后再一起讨论是用一个还是多个 CSS 文件。

6.2.1 提供CSS

用户访问包含 CSS 文件（与之相对的是使用行内 CSS）的网站时，浏览器首先要请求 CSS 文件，然后将其下载下来，再解析它们并应用恰当的样式。因此，我们需要尽可能地使需要下载的 CSS 文件缩小，以便提高加载速度。

拼接指的是将多个文件合并为一个文件的过程。拼接技术很常用，它通过减少需要下载的文件数量，降低页面加载时间。如果例 6-1 和例 6-2 的两个文件按照现有顺序进行拼接，将得到例 6-3 所示的结果。

例 6-1　headings.css 文件

```
/**
 * 该文件包含基本的标题元素的样式。
 */
h1 {
    color: #333;
    font-size: 24px;
    margin-bottom: 6px;
    margin-left: 12px;
    margin-right: 6px;
    margin-top: 12px;
}
```

例 6-2　lists.css 文件

```
/**
 * 该文件包含列表元素的样式。
 */
ul {
    list-style-type: none;
    padding-bottom: 12px;
    padding-left: 0;
    padding-top: 12px;
    padding-right: 0;
}
```

例 6-3　拼接后得到的 CSS 文件

```
/**
 * 该文件包含基本的标题元素的样式。
 */
h1 {
```

```
        color: #333;
        font-size: 24px;
        margin-bottom: 6px;
        margin-left: 12px;
        margin-right: 6px;
        margin-top: 12px;
    }
    /**
     * 该文件包含列表元素的样式。
     */
    ul {
        list-style-type: none;
        padding-bottom: 12px;
        padding-left: 0;
        padding-top: 12px;
        padding-right: 0;
    }
```

拼接操作功能强大，有了该方法，我们可以将大的 CSS 文件拆分为更小的文件（小文件可再重组为一个大文件，通过减少文件数量，终端用户不再需要下载多个文件，从而减少了页面加载时间）。如采用多个文件，相似的规则集则可以按照符合逻辑的方式组织到一起，因此更易于定位代码。此外，使用多个文件进行开发，更易于理解单个 CSS 文件的内容，而将所有 CSS 代码堆积到一个大文件中，则会增加理解代码的难度。

与 CSS 拼接相对应的一个概念是压缩。**压缩**是指从 CSS 文件中删除所有不必要的空格、注释和换行，而不改变代码行为的过程。例如，对例 6-1 和例 6-2 的代码进行拼接和压缩操作，将得到例 6-4 所示的代码。

例 6-4　压缩后的 CSS 文件

```
h1{color:#333;font-size:24px;margin-bottom:6px;margin-left:12px;margin-right:6px;
margin-top:12px}ul{list-style-type:none;padding-bottom:12px;padding-left:0;
padding-top:12px;padding-right:0}
```

> 压缩后的 CSS 通常只占一行；上述示例中的代码折行，是出于本书排版的需要。

请注意例 6-4 中，所有的空格和注释均已删除。空格和注释增加了用户需要下载的 CSS 文件的大小，删除文件中不必要的内容后，文件随之变小。你不要尝试按照压缩后得到的形式编写代码，因为出现错误难以查找，并且压缩形式的代码难以维护。

网上有很多拼接和压缩 CSS 的工具。当你准备做这一步时，请调研多种实现方式，使用符合自己需要的工具。也许你只需要使用简单的在线服务，把自己编写的 CSS 上传到他们的网站，网站再将压缩过后的 CSS 输出；也许你需要的是更复杂的工具，需要将其整合到开发过程中。

6.2.2 用单一的CSS文件进行开发

小型项目用一个 CSS 文件完全可以接受，操作起来也很简单。按照级联样式的工作原理安排 CSS 样式，合理地将 CSS 文件内容划分为几个大块，每个大块下再恰当安排小块内容，并合理添加注释：

```
/**
 * 通用样式
 * ---------------------------------------------
 */

/**
 * 基础样式
 * ---------------------------------------------
 */

 /* 基础样式：表单 */
 /* 基础样式：标题 */
 /* 基础样式：图像 */
 /* 基础样式：列表 */
 /* 基础样式：表格 */
 /* 等等 */

 /**
  * 组件样式
  * ---------------------------------------------
  */

 /* 组件样式：消息框 */
 /* 组件样式：按钮 */
 /* 组件样式：轮播框 */
 /* 组件样式：下拉框 */
 /* 组件样式：模态框 */
 /* 等等 */

 /**
  * 结构化样式
  * ---------------------------------------------
  */

 /* 结构化样式：结算区域的布局 */[1]
 /* 结构化样式：侧边栏的布局 */
 /* 结构化样式：主区域的布局 */
 /* 结构化样式：个人设置区域的布局 */
 /* 等等 */

 /**
  * 功能样式
  * ---------------------------------------------
  */
```

随着项目的成长，只用一个 CSS 文件，代码维护难度可能逐渐增大，当大到难以维护时，显然需要将其拆分为多个文件。

注 1：原文为“/* Structural Styles: Checkout Layout */”——译者注

6.2.3 用多个CSS文件进行开发

开发网站时使用多个 CSS 文件，各个文件所包含的样式可以分别服务于网站的某一部分功能，从而可以避免将 CSS 添加到不恰当的位置。开发时使用多个 CSS 文件，你的项目也许看起来像下面这样：

```
|-css/
| |-normalizing-styles
| |    |- normalize.css
| |
| |-base-styles
| |    |- forms.css
| |    |- headings.css
| |    |- images.css
| |    |- lists.css
| |    |- tables.css
| |    |- etc.
| |
| |-component-styles
| |    |- alerts.css
| |    |- buttons.css
| |    |- carousel.css
| |    |- dropdowns.css
| |    |- modals.css
| |    |- etc.
| |
| |- structural-styles
| |    |- layout-checkout.css
| |    |- layout-sidebar.css
| |    |- layout-primary.css
| |    |- layout-settings.css
| |    |- etc.
| |
| |- utility-styles
| |    |- utility.css
| |
| |- browser-specific-styles
| |    |-ie8.css
```

如果有这么多 CSS 文件，那么你不应该将其都添加到 HTML 文件中，因为请求的数量多了，将增加页面的加载时间。编写自动化任务，将其拼接为一个 CSS 文件，开发起来会更容易。

6.3 重构前审查CSS

从以下更高的角度来审查 CSS，将非常有助于重构：

- 所用到的属性列表
- 使用某一特定属性的声明块列表
- 使用的颜色数量
- 使用的最高和最低特指度
- 拥有最高和最低特指度的选择器
- 选择器的长度

CSS Dig 是 Google Chrome 浏览器的一款免费插件，你可以用它获取到以上信息。用 Chrome 浏览器访问 http://cssdig.com 安装该插件。安装完成后，其图标（图 6-1）将显示在已安装的浏览器插件旁边。



图 6-1：CSS Dig 插件图标

CSS Dig 的使用方法是访问目标网站，点击 CSS Dig 图标，然后弹出一个模态窗口，其中显示的是可以分析的 CSS 源文件和无法分析的文件（图 6-2）。



图 6-2：CSS Dig 模态窗口

点击模态窗口的“START DIGGING”按钮，CSS Dig 开始分析 CSS。这时将出现另一个含有两个页卡（Properties 和 Selectors）的模态窗口，它列出了这两个主题的详细信息（图 6-3）。



图 6-3：CSS Dig 属性页卡

从下面几节的学习中，你将发现 CSS Dig 是一个非常有价值的工具，能够帮你调整好 CSS 代码。

6.4　重构策略

本节探讨 CSS 的重构策略。但是要注意，条件允许的话，应该只对你能够维护的小块代码进行重构，并做到经常评审和发布。如果一次重构和发布大量代码，引入错误的风险更大，因为改动的内容更多。重构大量代码还会延缓新功能的编码，如果处理不当，在源文件版本控制系统合并代码也将更加复杂，甚至将引入错误。如果对小块代码进行重构，可能引发错误问题的改动将会更少，代码测试更加容易。

在第 3 章中我们学习了编写优秀 CSS 代码的多种技巧，但是在已经编写了大量代码的项目中却难以实现它们。我们的终极目标是运用这些理念，构建由通用样式、基础样式、组件、结构化样式和功能性样式组成的 CSS 代码库。接下来介绍的这些策略能够帮你实现每种理念，这样做可以鼓励从能够尽快部署的小改动开始重构。

6.4.1　保持规则集结构的一致性

保持规则集结构的一致性可以使开发更容易。请确定好声明块的格式和声明语句的顺序。每条声明语句可以各占一行，并尽可能按照字母顺序排列。按照上述方式修改现有的 CSS 或编写新的 CSS，可以保证规则集的一致性，从而降低风险。

6.4.2　删除僵尸代码

CSS 代码库不能留有僵尸代码，这一点真的很重要。**僵尸代码**是指存在但没有使用的代码。CSS 中的僵尸代码具体表现为以下几种形式：没有使用的声明块、重复的声明块和声明语句。

没有使用的声明块是指存在但从没有使用过的声明块，常由开发人员的疏忽导致；如果编写代码时思路不清晰或随着时间的推进发生了很多变化，那么难以跟踪所用样式。

重复的声明块是指没有必要的声明块，因为它们跟现有声明块雷同（例 6-5）。重复的声明块通常因复制和粘贴代码导致。

例 6-5　重复的声明块

```
h1 {
    font-size: 28px;
    font-weight: 900;
}

/* 重复的声明 */
h1 {
```

```
    font-size: 28px;
    font-weight: 900;
}
```

类似的，**重复的声明语句**（例 6-6）是指与同一声明块中的其他语句重复的声明语句。不同的规则集中声明语句重复的现象很常见，因为级联方法会指定样式起作用的顺序（虽然应尽可能将这些样式写到一起）。然而，我们应该避免同一规则集中存在重复的声明语句，因为最后出现的声明语句将起作用，而前面的则是多余的。

例 6-6 存在重复声明语句的声明块

```
h1 {
    font-size: 28px;
    font-weight: 900;
    font-weight: 900; /* 重复的声明 */
}

.section-title {
    font-size: 24px;
    /* 该声明并不算重复，因为它位于另一声明块之中，且作用于不同的选择器。
     */
    font-weight: 900;
}
```

僵尸代码非常有害，它增加了代码库的混乱程度，使其难以维护。修改代码时，你需要非常小心，不能破坏现有样式。如果存在大量僵尸代码，你不得不将大量时间浪费在对它们的理解上。除此之外，CSS 代码需要从服务器发送到终端用户的浏览器，若 CSS 文件中包含大量僵尸代码，下载时间将会延长，从而对用户体验带来破坏作用（尤其是网络连接较慢时）。

6.4.3 分离CSS和JavaScript

为元素添加样式的类和 ID 不应该在 JavaScript 中用作选择器，因为混用会增加更改它们的难度。CSS 和 JavaScript 应该尽早分离，因为类和 ID 很可能会调整，这可能会破坏 JavaScript 代码。

分离 CSS 和 JavaScript 很容易，搜索 JavaScript 代码，找到选取元素的位置，然后在选择器前面添加 `js-`，同时将该选择器添加到 HTML 代码中定义该元素的位置。

如果你使用某一 JavaScript 框架，请查阅该框架的文档找到元素的选取方式，然后在 JavaScript 代码中搜索选取元素的位置。如果你没有用框架，搜索 `document.getElement` 字符串应该能够找到选取元素的语句，因为原生 JavaScript 代码使用 `document.getElementById` 或 `document.getElementsByClassName` 选取元素。在 HTML 中查找元素很简单，搜索之前在 JavaScript 中使用的类名或 ID 即可。

6.4.4　分离基础样式

任何网站的基础样式都应该尽可能的简单，因为它们使用的是类型选择器。用 CSS Dig 插件分析网站样式后，Selectors 面板（图 6-4）给出了某一选择器的多种使用信息。

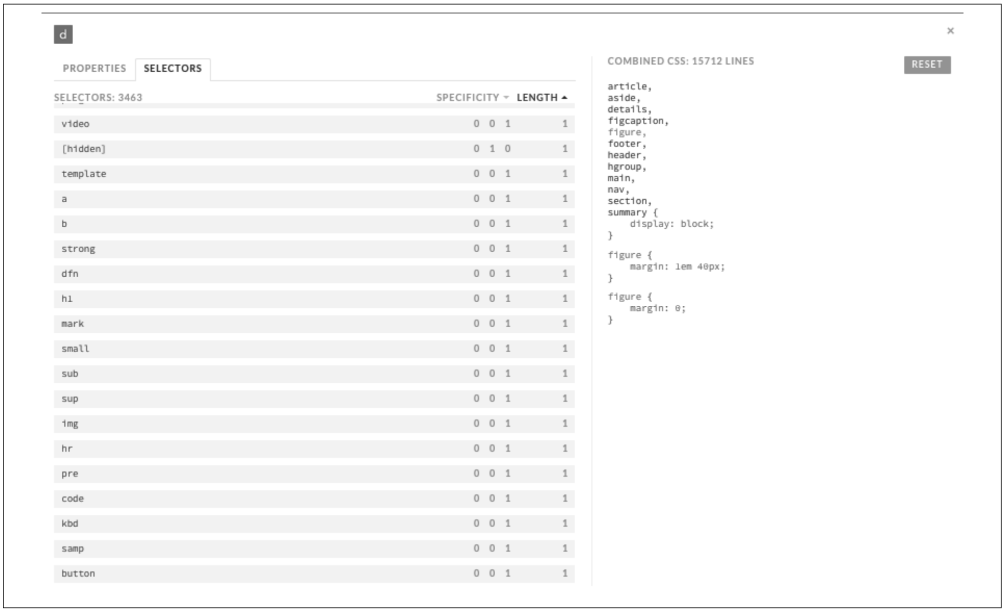


图 6-4：CSS Dig 的 Selectors 面板

将基础样式分为以下不同类别：

- 标题（<h1> – <h6>）
- 文本（例如：<p>、<fig> 和 <code>）
- 超链接（<a>）
- 列表（<dl>、 和 ）
- 表单（例如：<form>、<legend>、<fieldset>、<input> 和 <button>）
- 表格（例如：<table>、<thead>、<tbody>、<tfoot>、<tr> 和 <td>）
- 媒体（例如：<audio>、<object> 和 <video>）

分完类之后，你可以用 CSS Dig 寻找某一特定类别的选择器。找到选择器的使用频率和使用位置之后，你可以对它们进行比较，看看哪个属性更常用。如果一个类型选择器单独使用，且只用过一次，就可以放心将其删除。然而，如果一个类型选择器用过多次，请按以下步骤重构。

(1) 对于样式要予以重构的类型选择器，在基础样式中新建一条规则集。
(2) 从所有用到该类型选择器的地方找到最常用的属性，将其添加到新规则集中。
(3) 从其他规则集删除重复的属性，因为它们可以继承新定义的基础样式。

举个例子，假定我们用 CSS Dig 发现 `<h1>` 标签有多处定义，如例 6-7 所示，我们就可以对其重构。

例 6-7　`<h1>` 标签应用的多种样式

```
/* 文件：styles.css */

h1 {
    color: #000000;
    font-family: Helvetica, sans-serif;
    font-size: 32px;
    line-height: 1.2;
}

body > div > h1 {
    color: #000000;
    font-size: 32px;
    margin-bottom: 12px;
}

.section-condensed h1 {
    color: #000000;
    font-size: 16px;
}

.order-form h1 {
    font-family: Helvetica, sans-serif;
    color: #333333;
    text-decoration: underline;
}
```

第 1 步，新建一个规则集。

```
/* 文件：headings.css */

h1 {

}
```

第 2 步，将最常用的样式复制到新规则集中。

```
/* 文件：headings.css */

h1 {
    color: #000000;
    font-family: Helvetica, sans-serif;
    font-size: 32px;
    line-height: 1.2;
}
```

第 3 步，从规则集删除能够从基础样式继承的重复样式。

```
/* 文件：styles.css */
```

```
body > div > h1 {
    margin-bottom: 12px;
}

.section-condensed h1 {
    font-size: 16px;
}

.order-form h1 {
    color: #333333;
    text-decoration: underline;
}
```

如果进行如上改动之后页面样式出现问题，请用你最喜欢的浏览器开发者工具查看元素，然后核查 CSS 面板的样式以确定问题样式的源头。如果该问题难以修复，则可以使用精确程度更高的选择器添加临时样式来解决这个问题。但是要记得在后面添加一条注释，说明该处样式是临时样式以及为什么添加。继续重构的过程中，你将最终能够删除所有这些临时样式。

最后，详细检查未纳入基础样式的样式，确认其是否偏离它们应该发挥的作用。如果是的话，那么就可以将其删除。

6.4.5　删除冗余的ID

CSS 文件中特指度最高的选择器使用 ID。对拥有多个 ID 的选择器进行重构时，首先可以将最右侧 ID 左边的一切统统删除。因为同一个 ID 在一个网页最多只能使用一次，一个选择器包含多个 ID 实属冗余。例如，例 6-9 中选择器选择的元素与例 6-8 中相同，但是它的特指度却比较低。如果降低某一元素的特指度，其样式被更精确的选择器的样式覆盖，那么我们将无法降低该元素的特指度。遇到这种情况，重构时应该首先降低更精确的选择器的特指度。

例 6-8　选择器存在冗余 ID

```
#main-header > div > div > ul#main-menu {
    border-bottom: 1px solid #000000;
}
```

例 6-9　从选择器删除冗余 ID

```
#main-menu {
    border-bottom: 1px solid #000000;
}
```

6.4.6　将ID转化为类

删除冗余 ID 之后，剩余的 ID 可以转化为类。该过程虽然要花费一定时间，但是最终得到的 CSS 特指度更低、更易于复用。将 ID 改为类时，请记得使用意义明确的类名，切记不

要使用晦涩或过于精确的名称。

将 ID 转化为类后，你也许会发现由于特指度降低，一些元素的样式与预期不符。可能的修复方式是，找到具备较高特指度的选择器，降低它们的特指度，因为它们覆盖了其他特指度较低的选择器的样式。然而，如果解决某一特定问题需要改动大量的选择器，那么你最好不要将 ID 转化为类，而是等将更精确样式选择器的特指度降低之后，再来重构这部分代码。

6.4.7 区分功能性样式

功能性样式是唯一应该使用 `!important` 声明的样式。在你的 CSS 中搜索 `!important`，希望只能找到几处。若不得不使用 `!important` 的样式，且用途单一（如隐藏元素），应将其作为功能样式写到样式表的同一地方。本章前面我们讲过，在拼接 CSS 时功能样式应该置于 CSS 文件的底部，因此整合 CSS 文件时，一定要恰当安排功能性样式的位置。

对于非功能性样式而使用 `!important` 声明的，你应该用自己最喜欢的浏览器的开发者工具进行分析，弄清楚为什么使用该声明。如果经过分析，你认为不需要使用 `!important`，可以安全地将其删除。然而，如果 `!important` 用于覆盖继承来的样式，可暂时予以保留，并以注释的形式说明使用该声明的原因。以后对起覆盖作用的 `!important` 声明进行重构时，再重新审查该样式并进行重构，然后就可以安全地将其删除了。

6.4.8 定义可复用组件

重构 CSS 时，定义可复用组件是最令人畏惧的任务之一，因为我们通常会担心无法一次成功。然而，这种担心是多余的，因为如果你无法保证第 1 次重构时就做到 100% 正确，那么以后也可以再次研究该组件并将其改好。

你可以从经常重复使用的界面模式（例如：选项卡）着手，花点时间调研网站什么地方用到了该模式。请记录该模式的各种变体，并判断它们是合乎规范的，还是由于不一致的 CSS 而导致的。你可以按照第 4 章中“组件样式”一节的指南构建可复用组件，然后用新的组件代码替换之前所有用到该模式的地方。

定义可复用组件有助于删除重复的 CSS。删除多余的 ID 后，通常余下的那个 ID 用于为元素添加样式。可以用作可复用组件的界面模式，第 1 次实现时因为是从头开始编码，不存在代码重复问题，但是当定义相似却略为不同的模式时，势必需要从第 1 次实现该组件的地方复制代码，并进行修改以满足新的、略为不同的界面模式的需求，那么代码重复问题就会出现。使用相同的组件样式可以防止出现该问题，但是若要重新定义组件的样式，则要把样式添加到容器组上或使用结构化样式。

6.4.9 删除行内CSS和过于模块化的类

删除行内 CSS 和过于模块化的类应该同时进行，因为它们在本质上是相同的，只不过用 `style` 属性添加的行内 CSS 其特指度更高，除非用 `!important` 声明覆盖了行内样式。这两个方面都应该晚些重构。

如果按照本章所讲的重构顺序进行重构，那么到目前为止你：

- 正在朝保持 CSS 在结构上的一致性而努力
- 有更少的僵尸代码
- 分离了 CSS 和 JavaScript
- 建立了基础样式
- 通过删除多余的 ID 和将 ID 转化为类，降低了特指度最高的选择器
- 将功能性样式单独放在一起，减少了 `!important` 声明的使用
- 定义了可复用组件

完成以上重构，你可以迁移行内样式了——如果代码中还有残留的！如果在这之前删除行内样式，你可能需要将其放到样式表最后临时添加的类之中，同时为了保持它们的特指度，你很可能要使用 `!important` 声明。如果等到重构完其他项后再迁移行内 CSS，你只需要搜索 HTML 文件找到行内样式，如果不再需要它们，可直接将其删除或根据需要将其移植到样式表的合适位置。再次提醒，有悖于基础样式和组件样式的行内样式，应予以仔细调查，以确定其是否是因为不一致的设计或编码而添加的。如果确实如此，可以将行内样式安全删除。如果样式上的变异是必要的，则可以尝试通过为容器添加样式来达到修改元素样式的目的。

6.4.10 隔离面向特定浏览器的CSS样式

浏览器之间存在差异，因此为了突破浏览器的限制，我们会在 CSS 中使用一些特定的“黑技术”，而这些代码很容易污染其他 CSS，因此我们需要对其进行区分。但是在隔离之前，请记得查看网站的流量来源，以判断是否能够放弃支持这些浏览器。删除浏览器特定样式比起重构代码为其找到合适的位置更易于操作，结果也更令人满意。如果无法放弃对一款浏览器的支持，那么你只需要使用第 4 章所讲的条件注释专门针对该浏览器为某一元素添加样式即可。

6.5 评估重构是否成功

我们学习了以上重构技巧之后，理解如何评定重构的成功与否非常重要，以便为重构定下实际可以操作的目标。下面是重构完成之后如何评价是否成功的一些想法。其中一些建议，比如检查文件的大小，在重构之前也应要做，以便与重构之后进行比较。

6.5.1 你的网站崩溃了吗

判断代码重构之后成功与否的首要的、最明显的方式是确认网站的行为是否倒退。再次提醒，重构是指在不改变代码行为的前提下，重写代码使其更加简洁。在第 5 章中，我们讨论了手动测试网站在多个浏览器的表现，以及通过比较截图的方式进行了视觉回归测试。如果经过彻底的视觉效果检测之后没有发现视觉效果方面的问题，那么接下来你需要考虑其他方面。

1. 低耦合度

高质量的 CSS 是指 CSS 与使用它的 HTML 结构分离开来。按照第 4 章所讲的技巧，通过创建可复用组件和用容器包裹组件，你可以实现 CSS 和 HTML 的分离。虽然一定程度的耦合度一直存在，但是还是要避免使用过于复杂的选择器。CSS Dig 这类工具能够帮你嗅探出复杂和过于精确的选择器，你需要将其改成更为宽泛的。审计你的网站并检查你的选择器以找出哪些改进是必要的。

2. 低特指度

第 2 章讲过 CSS 特指度和规则集顺序决定着元素所应用的样式。我们在第 4 章也学习了降低整体 CSS 特指度的策略。选择器的特指度指标可用于度量代码库的特指度，以判断代码库是否包含大量高特指度的选择器，这些选择器将增加代码的维护成本。CSS Dig 这类工具可以按照选择器的特指度为其排序。

3. 更少的文件数量和更小的文件

本章前面讲解了应该向终端用户提供拼接和压缩过的 CSS 文件——拼接可以减少需要下载的文件数量，压缩可以删除多余字符以减小文件体积。这两种处理方法都能提高下载速度，进而减少页面加载时间。如要衡量这一指标，你可以分别统计并比较重构前后所有文件的大小。

6.5.2 UI bug数

一旦你开始重构 CSS 并遵循编码标准，因代码凌乱或代码重复而引入的 UI bug 数量应该会减少。第 5 章讲过实现 UI 模式库和用截图监控视觉效果的变化，能较好地保证用户界面是我们用经过全面测试的代码构建的，其视觉效果一直处于我们的监控之中。软件 bug 是不可避免的（网站开发人员可能引入 bug，浏览器厂商可能引入浏览器问题），但是使用模式库和执行视觉效果测试百试不爽，它们能够帮你更快地检测和诊断这些问题。

6.5.3 减少开发和测试时间

你将 CSS 以符合逻辑的方式分成多个文件，确立了编码标准，并创建了用户界面模式库之后，应该能够较以往更快地构建和维护用户界面了。该项指标因网站而异，因为有些用户界面比起其他的更为复杂，开发时间相应更长。但对于大多数情况，你还是能够判断重构

是否降低了开发时间，提高了工作流的效率。

除了降低开发时间，如果你能熟练使用正确的工具，你也许还注意到你可以更快地测试界面。再次提醒，第 5 章所讲的各种测试方法应该能够提升你测试界面的速度和自信心。

6.6 总结

本章讨论了 CSS 的组织方式、代码的重构策略和衡量重构成功与否的各项指标。切记这些策略要经常应用而不是只用一次，以便我们能以更小和更可控的模块发布这些改动。当你开始实现本章介绍的策略时，你已经开始构建遵循优秀架构所有原则的 CSS 代码库了。

附录

normalize.css

```
/*! normalize.css v5.0.0 | MIT License | github.com/necolas/normalize.css */

/**
 * 1. 修改在所有浏览器中的默认字体（自定义）。
 * 2. 纠正在所有浏览器中的行高。
 * 3. 在Windows Phone和iOS设备调整屏幕方向后，防止字体大小发生改变。
 */

/* 文档
   =================================================================== */

html {
  font-family: sans-serif; /* 1 */
  line-height: 1.15; /* 2 */
  -ms-text-size-adjust: 100%; /* 3 */
  -webkit-text-size-adjust: 100%; /* 3 */
}

/* 各分区
=================================================================== */

/**
 * 清除在所有浏览器中的margin（自定义）。
 */

body {
  margin: 0;
}

/**
 * 增加在IE 9-浏览器中的正确显示方式。
 */
```

```
article,
aside,
footer,
header,
nav,
section {
  display: block;
}

/**
 * 纠正section和article区块元素中的h1元素在Chrome、Firefox和Safari浏览器中的font-size
   和margin。
 */

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

/* 为内容分组
   ================================================================== */

/**
 * 增加在IE 9-浏览器中的正确显示方式。
 * 1. 增加在IE浏览器中的正确显示方式。
 */

figcaption,
figure,
main { /* 1 */
  display: block;
}

/**
 * 增加在IE 8浏览器中的正确margin值。
 */

figure {
  margin: 1em 40px;
}

/**
 * 1. 增加在Firefox浏览器中应显示的盒子尺寸。
 * 2. 在Edge和IE浏览器中应用的溢出样式。
 */

hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}

/**
 * 1. 纠正在所有浏览器中字体大小的继承和缩放比例。
```

```
 * 2. 纠正所有浏览器对以em为单位表示字体大小所产生的古怪效果。
 */

pre {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/* 文本层级的语义
   ================================================================== */
/**
 * 1. 清除IE 10浏览器为激活的链接添加的灰色背景。
 * 2. 清除在iOS 8+和Safari 8+系统中为链接添加的下划线之间的间隔。
 */

a {
   background-color: transparent; /* 1 */
   -webkit-text-decoration-skip: objects; /* 2 */
}

/**
 * 所有浏览器中，当链接处于激活或鼠标悬浮状态的聚焦状态时，清除链接的轮廓（自定义）。
 */

a:active,
a:hover {
  outline-width: 0;
}

/**
 * 1. 清除在Firefox 39-浏览器中的下边框。
 * 2. 增加在Chrome、Edge、IE、Opera和Safari浏览器中正确的文本修饰。
 */

abbr[title] {
  border-bottom: none; /* 1 */
  text-decoration: underline; /* 2 */
  text-decoration: underline dotted; /* 2 */
}

/**
 * 防止在Safari 6浏览器中，因下一条样式规则而重复应用border属性。
 */

b,
strong {
  font-weight: inherit;
}

/**
 * 增加在Chrome、Edge和Safari浏览器中正确的字体粗细样式。
 */

b,
strong {
```

```
  font-weight: bolder;
}

/**
 * 1. 纠正在所有浏览器中字体大小的继承和缩放比例问题。
 * 2. 纠正所有浏览器对以em为单位表示字体大小所产生的古怪效果。
 */

code,
kbd,
samp {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/**
 * 增加在Android 4.3-中正确的字体样式。
 */

dfn {
  font-style: italic;
}

/**
 * 增加在IE9-浏览器中正确的背景色和文本的颜色。
 */

mark {
  background-color: #ff0;
  color: #000;
}

/**
 * 增加在所有浏览器中正确的字体大小。
 */

small {
  font-size: 80%;
}

/**
 * 防止sub和sup元素影响在所有浏览器中的行高。
 */

sub,
sup {
  font-size: 75%;
  line-height: 0;
  position: relative;
  vertical-align: baseline;
}

sub {
  bottom: -0.25em;
}
```

```
sup {
  top: -0.5em;
}

/* 嵌入的内容
   ========================================================================== */

/**
 * 增加在IE 9-浏览器中的正确显示方式。
 */

audio,
video {
  display: inline-block;
}

/**
 * 增加在iOS 4-7浏览器中的正确显示方式。
 */

audio:not([controls]) {
  display: none;
  height: 0;
}

/**
 * 清除IE 10-浏览器中包裹在链接内部的图像的边框。
 */

img {
  border-style: none;
}

/**
 * 在IE浏览器中隐藏溢出的部分。
 */

svg:not(:root) {
  overflow: hidden;
}

/* 表单
   ========================================================================== */

/**
 * 1. 修改在所有浏览器中的字体样式（自定义）。
 * 2. 清除Firefox和Safari浏览器中的margin。
 */

button,
input,
optgroup,
select,
textarea {
```

```
  font-family: sans-serif; /* 1 */
  font-size: 100%; /* 1 */
  line-height: 1.15; /* 1 */
  margin: 0; /* 2 */
}

/**
 * 在IE浏览器中显示超出的部分。
 * 1. 在Edge浏览器中显示超出的部分。
 */

button,
input { /* 1 */
  overflow: visible;
}

/**
 * 在Edge、Firefox和IE浏览器中，清除对文本转换的继承。
 * 1.在Firefox浏览器中，清除对文本转换的继承。
 */

button,
select { /* 1 */
  text-transform: none;
}

/**
 * 1. 在Android 4系统中，防止WebKit的一个bug（下面第2点所描述的）破坏原生的audio
 *    和video控件。
 * 2. 纠正在iOS系统和Safari浏览器中无法为可点击类型添加样式的问题。
 */

button,
html [type="button"], /* 1 */
[type="reset"],
[type="submit"] {
  -webkit-appearance: button; /* 2 */
}

/**
 * 在Firefox浏览器中，清除内部border和padding。
 */

button::-moz-focus-inner,
[type="button"]::-moz-focus-inner,
[type="reset"]::-moz-focus-inner,
[type="submit"]::-moz-focus-inner {
  border-style: none;
  padding: 0;
}

/**
 * 根据前面的规则，将按钮聚焦样式恢复为未设置前的。
 */
```

```
button:-moz-focusring,
[type="button"]:-moz-focusring,
[type="reset"]:-moz-focusring,
[type="submit"]:-moz-focusring {
  outline: 1px dotted ButtonText;
}

/**
 * 修改在所有浏览器中的border、margin和padding（自定义）。
 */

fieldset {
  border: 1px solid #c0c0c0;
  margin: 0 2px;
  padding: 0.35em 0.625em 0.75em;
}

/**
 * 1. 纠正在Edge和IE浏览器中文本换行异常的问题。
 * 2. 纠正在IE浏览器中fieldset元素color属性不能正确继承的问题。
 * 3. 清除padding，开发者将fieldset元素在所有浏览器中的padding值都清零时，不至于出错。
 */

legend {
  box-sizing: border-box; /* 1 */
  color: inherit; /* 2 */
  display: table; /* 1 */
  max-width: 100%; /* 1 */
  padding: 0; /* 3 */
  white-space: normal; /* 1 */
}

/**
 * 1. 增加在IE 9-浏览器中的正确显示方式。
 * 2. 增加Chrome、Firefox和Opera浏览器中正确的垂直对齐方式。
 */

progress {
  display: inline-block; /* 1 */
  vertical-align: baseline; /* 2 */
}

/**
 * 清除IE浏览器中默认的垂直滚动条。
 */

textarea {
  overflow: auto;
}

/**
 * 1. 增加IE 10-浏览器中正确的盒子尺寸类型。
 * 2. 在IE 10-浏览器中清除padding。
 */
```

```
[type="checkbox"],
[type="radio"] {
  box-sizing: border-box; /* 1 */
  padding: 0; /* 2 */
}

/**
 * 纠正在Chrome浏览器中增加和减少按钮组件的光标样式。
 */

[type="number"]::-webkit-inner-spin-button,
[type="number"]::-webkit-outer-spin-button {
  height: auto;
}

/**
 * 1. 纠正Chrome和Safari浏览器中的古怪样式。
 * 2. 纠正在Safari中的轮廓样式。
 */

[type="search"] {
  -webkit-appearance: textfield; /* 1 */
  outline-offset: -2px; /* 2 */
}

/**
 * 清除macOS系统下Chrome和Safari浏览器的内部padding和cancel button（取消按钮）样式。
 */

[type="search"]::-webkit-search-cancel-button,
[type="search"]::-webkit-search-decoration {
  -webkit-appearance: none;
}

/**
 * 1. 纠正无法在iOS系统和Safari浏览器中为可点击类型的元素应用样式的问题。
 * 2. 在Safari浏览器中将font属性值改为inherit。
 */

::-webkit-file-upload-button {
  -webkit-appearance: button; /* 1 */
  font: inherit; /* 2 */
}

/* 交互
   ================================================================== */

/*
 * 增加在IE 9-浏览器中的正确显示方式。
 * 1. 增加在Edge、IE和Firefox浏览器中的正确显示方式。
 */

details, /* 1 */
menu {
  display: block;
```

```
}

/*
 * 增加在所有浏览器中的正确显示方式。
 */

summary {
  display: list-item;
}

/*脚本
   ================================================================== */

/**
 * 增加在IE 9-浏览器中的正确显示方式。
 */

canvas {
  display: inline-block;
}

/**
 * 增加在IE浏览器中的正确显示方式。
 */

template {
  display: none;
}

/* 隐藏
   ================================================================== */

/**
 * 增加在IE 10-浏览器中的正确显示方式。
 */

[hidden] {
  display: none;
}
```

作者简介

Steve Lindstrom 早在 1999 年出于个人爱好开发了自己的第一个网站，那时他还在中学读书。后来他赴佛罗里达州墨尔本市求学，从佛罗里达理工学院获得了计算机科学学士学位。Steve 曾在国防、旅游科技领域从事软件开发工作，最近开始涉足电子商务领域。工作之余，他喜欢学习烹饪和喝咖啡。

封面说明

本书封面上的动物是非洲椰子狸（Nandinia binotata），也称双斑狸。这是一种跟黄鼠狼和獴非常相近的小型杂食性哺乳动物，主要生活在东非，但是在非洲大陆中西部地区也有分布。它们喜欢栖息在低地森林或赤道附近的丛林之中。

非洲椰子狸腿短、尾巴长，棕黑色的皮毛上夹杂着黑斑。这种动物有几分像家猫，并且比起其他灵猫科动物更像一些，因此，它们是 Nandinia 属的唯一成员，平均体重为 3~5 磅（1 磅 =0.454 千克）。它们是夜行性动物，白天绝大部分时间都栖息在大树的隐秘处，夜间觅食。虽然非洲椰子狸的食物以水果为主，但是它们也捕食昆虫、蜥蜴、鸟类、蝙蝠和小型啮齿动物。

非洲椰子狸一年繁殖两次，分别在 5 月和 10 月（这两个月份可获取的食物通常更为丰富）。幼崽出生以后，母兽的乳腺会分泌一种橙黄色液体来染黄自己的腹部和幼崽的皮毛。母兽这样做的原因我们尚未完全弄清，不过它们可以以此作为自己暂不能交配的信号，或警告雄兽远离自己，以免伤害自己的孩子。

灵猫香指的是这类动物的分泌物，灵猫科动物凭借它划定领土范围和寻找配偶。灵猫香经稀释之后可用作香料，人们使用这种香料已经有成百上千年的历史了。虽然如今很多产品使用人工合成的灵猫香，但是为了获取灵猫科几种动物的肉和气味腺，有人仍在非法捕捉它们。

O’Reilly 图书封面上的很多动物都是濒危物种，它们对整个世界都很重要。如果你想为保护动物做些贡献，请访问 animals.oreilly.com。

封面照片来自 Lydekker 的 *Royal Natural History*。

O'REILLY®

CSS重构：样式表性能调优

CSS对于现代网站的用户体验至关重要，其地位不亚于决定着网站外形的HTML和让网站动起来的JavaScript。本书作为CSS代码重构指南，不仅展示了如何编写结构合理的CSS，以构建响应式、易于使用的网站，还介绍了如何用重构方法创建可读性更强和更易于维护的CSS代码。不论你是刚开始开发自己的第一个CSS项目或是清理现有项目的代码，本书提供的多种宝贵方法都可以帮你建设一个符合优秀建构设计原则的CSS代码库。

- 了解什么是代码重构及其与CSS之间的关系
- 探索Web浏览器如何使用级联方法决定为哪个元素应用什么样式
- 编写可预测、易维护和可扩展的CSS，提升代码复用能力
- 理清不同类型的CSS样式及其使用场景
- 确定对哪些浏览器和设备进行测试，以维护好CSS
- 学习如何合理组织样式，重构CSS和评估重构效果

Steve Lindstrom，经验丰富的软件工程师，在中学读书时就出于个人爱好开发了自己的第一个网站。曾为国防、旅游科技和电子商务领域开发过相关软件。

CSS / HTML

封面设计：Karen Montgomery 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机 / Web开发 / CSS

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-115-46978-6

ISBN 978-7-115-46978-6
定价：39.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks